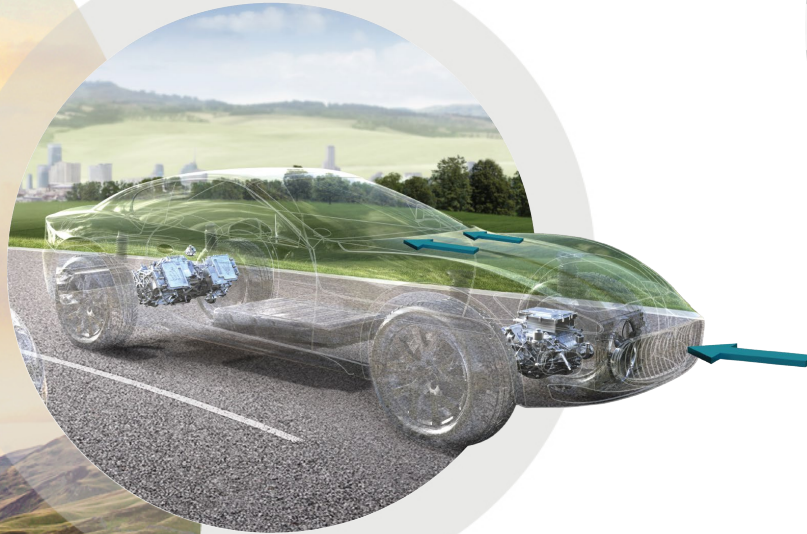




The Power of Python in KULI

Chris Plakolm, Engineer KULI Methods, Magna &
Christoph Stroh, Manager KULI Software
Methods, Magna

ECS Simulation Conference, 10th-11th May 2023
Linz, Austria

Content

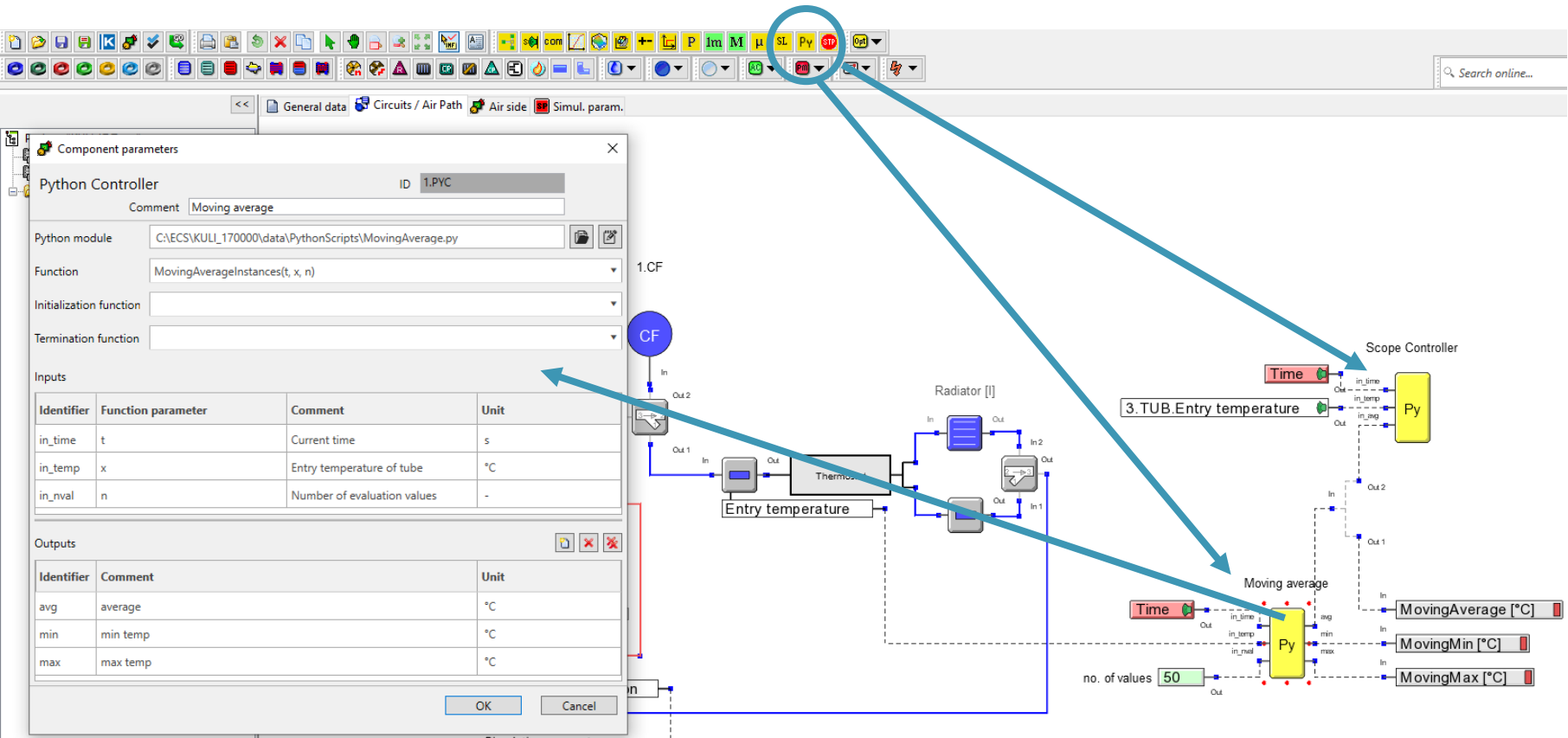


- Functionality of Python Controller component
- Practical examples
 - Replace complex signal paths
 - Data analysis – moving average
 - Transient optimization
 - Plotting with Python
 - Get operation points from Excel[®]
 - Python-Simulink interface
- Conclusion

An abstract background featuring a dark field with vibrant red and blue light trails streaking across it. Overlaid on these are complex wireframe structures in blue and red, resembling architectural or technical designs.

Functionality of Component

Python Controller GUI



Component Parameters

- Python module
 - Enter/browse for path to target .py script
 - default folder is: ..\data\PythonScripts
- Function
 - dropdown list of all functions in target .py script (with at least one input)
 - is executed in every call to the PYC
- Inputs
 - Number of inputs is recognized from the function selected above



The screenshot shows a 'Component parameters' dialog box for a 'Python Controller' with ID '1.PYC'. It includes fields for a comment, a Python module (with a selection prompt), a function (dropdown), an initialization function (dropdown), and a termination function (dropdown). Below these are sections for 'Inputs' and 'Outputs', each with a table for defining parameters. The 'Inputs' table has columns for Identifier, Function parameter, Comment, and Unit. The 'Outputs' table has columns for Identifier, Comment, and Unit. At the bottom are 'OK' and 'Cancel' buttons.

Identifier	Function parameter	Comment	Unit
------------	--------------------	---------	------

Identifier	Comment	Unit
------------	---------	------

Component Parameters

- Inputs (and outputs)
 - Identifier:
are shown in Circuits / Air Path UI
 - Function Parameter:
respective arguments of Python function
(cannot be changed)
 - Comment:
any text
 - Unit:
select from list of units
must be selected for each input



The screenshot shows the 'Component parameters' dialog box for a 'Python Controller'. The ID is '1.PVC'. The Python module is 'CAECS\KULL_170000\data\PythonScripts\PythonTestScript.py'. The function is 'TestFunction(Input_1, Input_2, Input_3)'. The initialization and termination functions are empty. Below these are sections for 'Inputs' and 'Outputs'.

Inputs

Identifier	Function parameter	Comment	Unit
IN_1	Input_1		
IN_2	Input_2		
IN_3	Input_3		

Outputs

Identifier	Comment	Unit
------------	---------	------

At the bottom are 'OK' and 'Cancel' buttons.

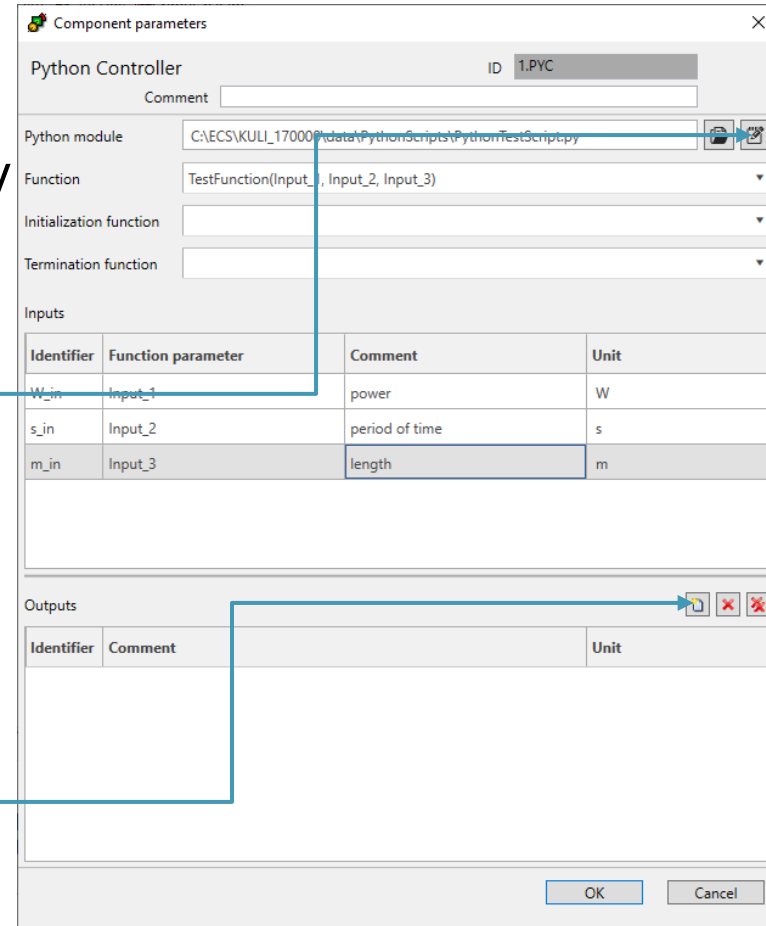
Component Parameters

- Outputs

- Number of outputs is not set automatically
- Multiple outputs are possible
 - min: 0 outputs
 - max: depends on script → open script

```
PythonScripts > PythonTestScript.py > ...
1  """This is a test script"""
2
3  def TestFunction(Input_1, Input_2, Input_3):
4      """This is a test function"""
5      return Input_1 * Input_2 , Input_3 * Input_3 + 1
```

- outputs = return values
(separated by comma, from left to right)
- Need to be created one by one
- Units are required



Component parameters

Python Controller ID: 1.PVC

Comment:

Python module: CAECS\KULL_170009\data\PythonScripts\PythonTestScript.py

Function: TestFunction(Input_1, Input_2, Input_3)

Initialization function:

Termination function:

Inputs

Identifier	Function parameter	Comment	Unit
W_in	Input_1	power	W
s_in	Input_2	period of time	s
m_in	Input_3	length	m

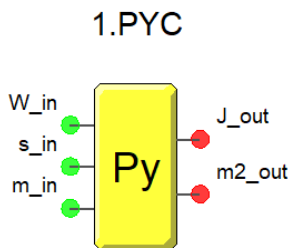
Outputs

Identifier	Comment	Unit
------------	---------	------

OK Cancel

Component Parameters

- Optional functions
 - dropdown list of all functions with no input arguments
 - Initialization function:
 - called at the start of the simulation (before the first iteration)
 - Termination function:
 - called at the end of the simulation (after the last iteration)



Component parameters

Python Controller ID 1.PYC

Comment

Python module CAECS\KULL_170000\data\PythonScripts\PythonTestScript.py

Function TestFunction(Input_1, Input_2, Input_3)

Initialization function

Termination function

Inputs

Identifier	Function parameter	Comment	Unit
W_in	Input_1	power	W
s_in	Input_2	period of time	s
m_in	Input_3	length	m

Outputs

Identifier	Comment	Unit
J_out	energy	J
m2_out	area	m ²

OK Cancel

Practical Examples

... without too much coding

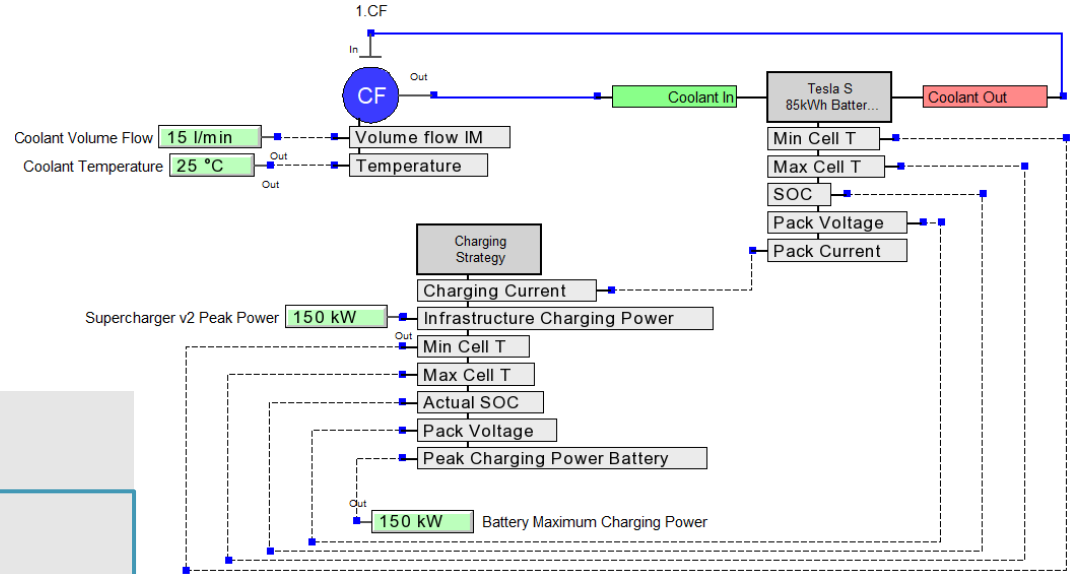
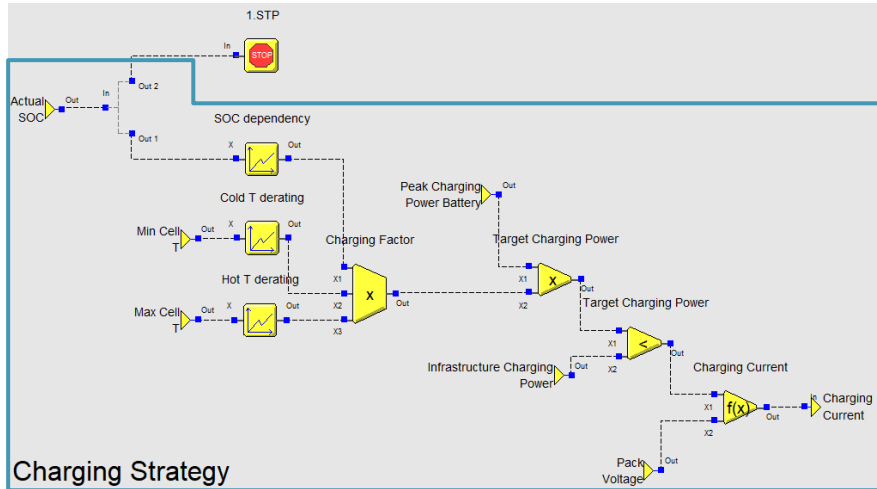
An abstract background featuring a dense network of glowing red and blue lines that form complex, overlapping geometric patterns, resembling a signal path or a data network.

Replace complex signal paths

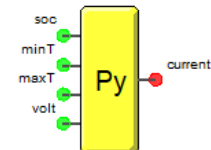
Battery Cooling for Tesla Model S



Replace subsystem
“Charging Strategy”
by a Python Controller



Charging Strategy

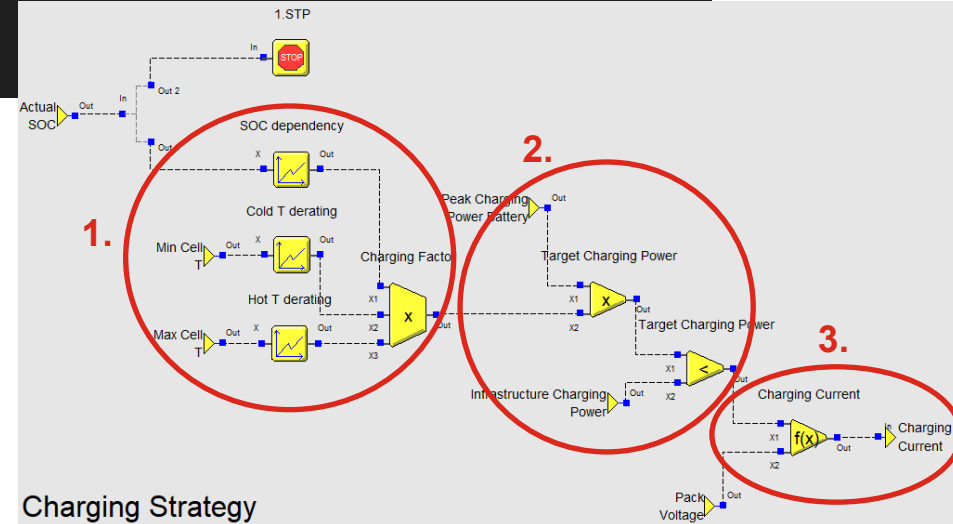


The Python Script

PythonScripts > charging_strategy.py > ...

```
28
29 # define function for Python Controller
30 def CalculateChargingCurrent(ActualSOC, MinCellT, MaxCellT, PackVoltage):
31     '''Given the actual (current) state of charge [%],
32     the minimum and maximum temperature of the cell [K], respectively,
33     and the voltage of the battery pack [V],
34     this function returns the charging current of the battery pack [A].'''
35
36     1. charging_factor = SOCdependency(ActualSOC) * coldTempDerating(MinCellT) * hotTempDerating(MaxCellT) # [-]
37     2. target_charging_power = min(CP_PEAK * charging_factor, CP_INFRASTRUCTURE) # [kW]
38
39     # (1000*[kW] = [W]) / V = [A]
40     3. return - 1000*target_charging_power/PackVoltage # [A]
```

1. product of characteristic curves
2. minimum function
3. custom function (units!)



The Python Script



constants

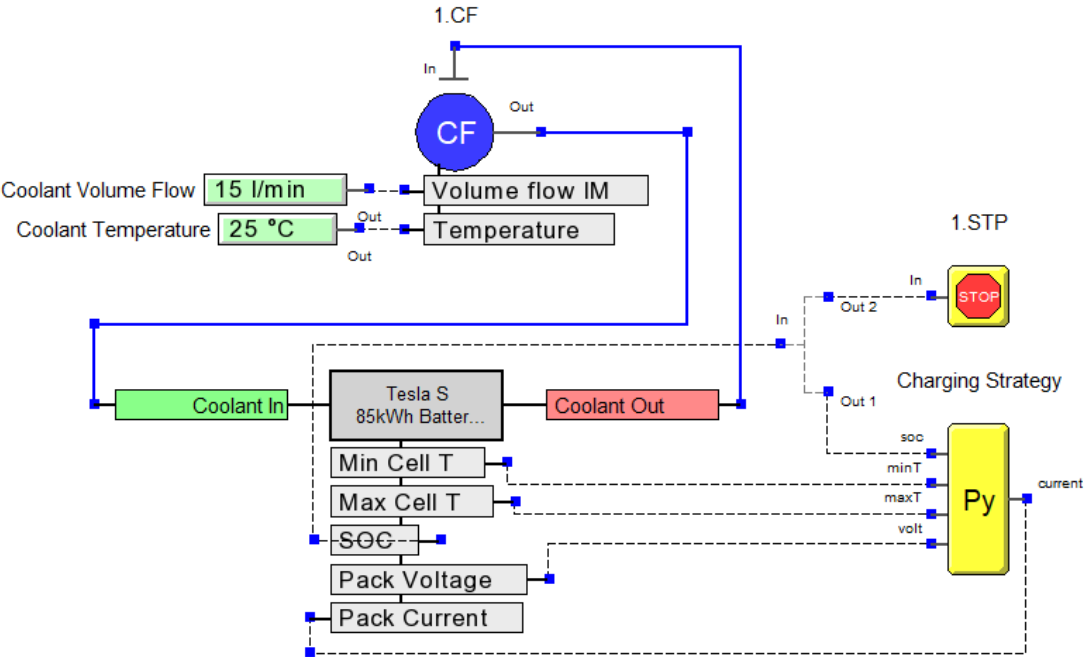
data for
characteristic
curves

mappings

(linear interpolation)

```
PythonScripts > charging_strategy.py > ...
2  from numpy import interp
3
4  # define constants: charging power CP [kW]
5  CP_INFRASTRUCTURE = 150
6  CP_PEAK = 150
7
8  # define arrays for mappings
9  soc_dependency_x = [0, 10, 30, 37, 44, 50, 56, 65, 75, 80, 100]
10 soc_dependency_y = [.78, .78, .71, .64, .58, .51, .45, .38, .33, .29, .13]
11 coldT_derating = [0, 15]
12 hotT_derating = [35, 45]
13
14 # define auxiliary functions for mappings
15 def SOCdependency(SOC):
16     '''Characteristic curve: input in [%], output in [-]'''
17     return interp(SOC, soc_dependency_x, soc_dependency_y)
18 def coldTempDerating(cold_temp):
19     '''Derating factor for cold temperatures: input in [°C], output in [-]'''
20     return interp(cold_temp, coldT_derating, [0, 1])
21 def hotTempDerating(hot_temp):
22     '''Derating factor for cold temperatures: input in [°C], output in [-]'''
23     return interp(hot_temp, hotT_derating, [1, 0])
```

New Model



Component parameters

Python Controller

ID1.PVC

CommentCharging Strategy

Python moduleC:\ECS\KULI_170000\data\PythonScripts\charging_strategy.py

FunctionCalculateChargingCurrent(ActualSOC, MinCellIT, MaxCellIT, PackVoltage)

Initialization function

Termination function

Inputs

Identifier	Function parameter	Comment	Unit
soc	ActualSOC		%
minT	MinCellIT		°C
maxT	MaxCellIT		°C
volt	PackVoltage		V

Outputs

Identifier	Comment	Unit
current	Pack Current	A

OKCancel

Data Analysis – Moving Average

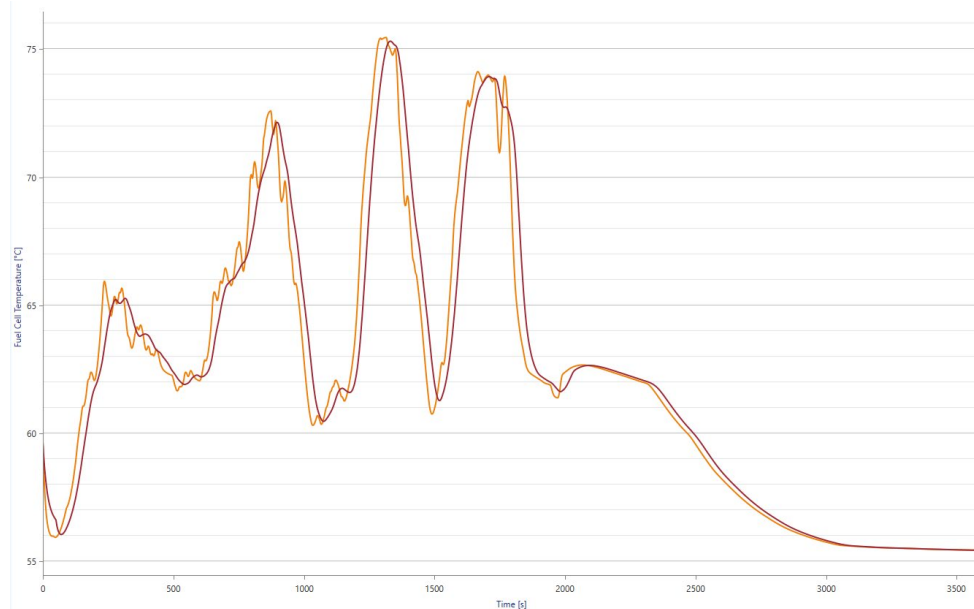
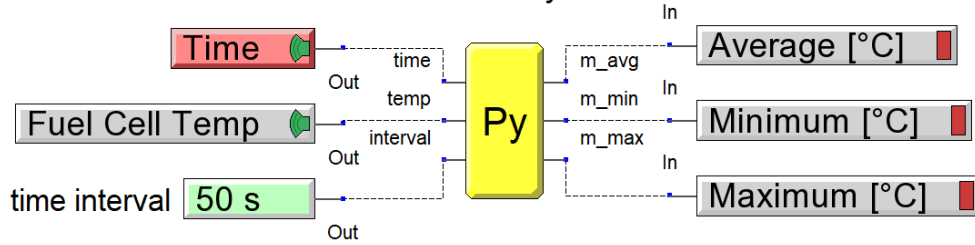
Moving Average



- Calculation of the arithmetic mean of data points in a subset of the full data set.
- Data set is continuously updated
 - size of data set increases with time
- Subset consists of the latest few entries of full data set
 - either contains the latest n entries
 - or contains points that have been added during the last `delta_t` seconds
- Analogous for the calculation of the moving minimum and maximum

The Python Controller

Data Analysis



 Component parameters ✕

Python Controller

ID 3.PYC

Comment

Python module

C:\ECS\KULI_170000\data\PythonScripts\MovingAverage.py

Function

MovingAverageInstances(t, x, n)

Initialization function

Termination function

Inputs

Identifier	Function parameter	Comment	Unit
time	t	current time	s
temp	x	temperature	°C
samples	n	number of samples	-

Outputs

Identifier	Comment	Unit
m_avg	moving average	°C
m_min	moving minimum	°C
m_max	moving maximum	°C

OK

Cancel

Time Step vs. Iteration



- PYC is called (i.e. Python function is executed) in each Iteration. A time step has multiple Iterations.
- In function: differentiate between new time step/iteration
 - return value: „remember“ n latest inputs (store in global list `data`), calculate sum over those inputs, divide the sum by n .
 - If new time step: remove oldest input from and add current input to the global list `data`.
 - if new iteration: update latest input in global list `data`.
- Track time steps/iterations with global variable `Lasttime`
 - Define globally as `Lasttime = -1`

Time Step vs. Iteration



```
def MovingAverageN(t, x, n):  
    """Given the current time t, an input x and an integer n, return the moving average of the last n inputs x."""  
  
    global Lasttime  
    global data  
  
    numVal = int(max(1, n))    # max size of data list  
  
    if t > Lasttime:           # new time step  
        data.append(x)         # append latest input  
        data = data[-numVal:]  # keep latest numVal inputs  
    else:                      # new iteration, same time step  
        data[-1] = x           # update latest input  
  
    m_avg = sum(data)/len(data)  
    m_min = min(data)  
    m_max = max(data)  
  
    return m_avg, m_min, m_max
```

An abstract background image featuring a blue wireframe grid structure, possibly representing a building or a complex system, overlaid with vibrant red light streaks that create a sense of motion and energy.

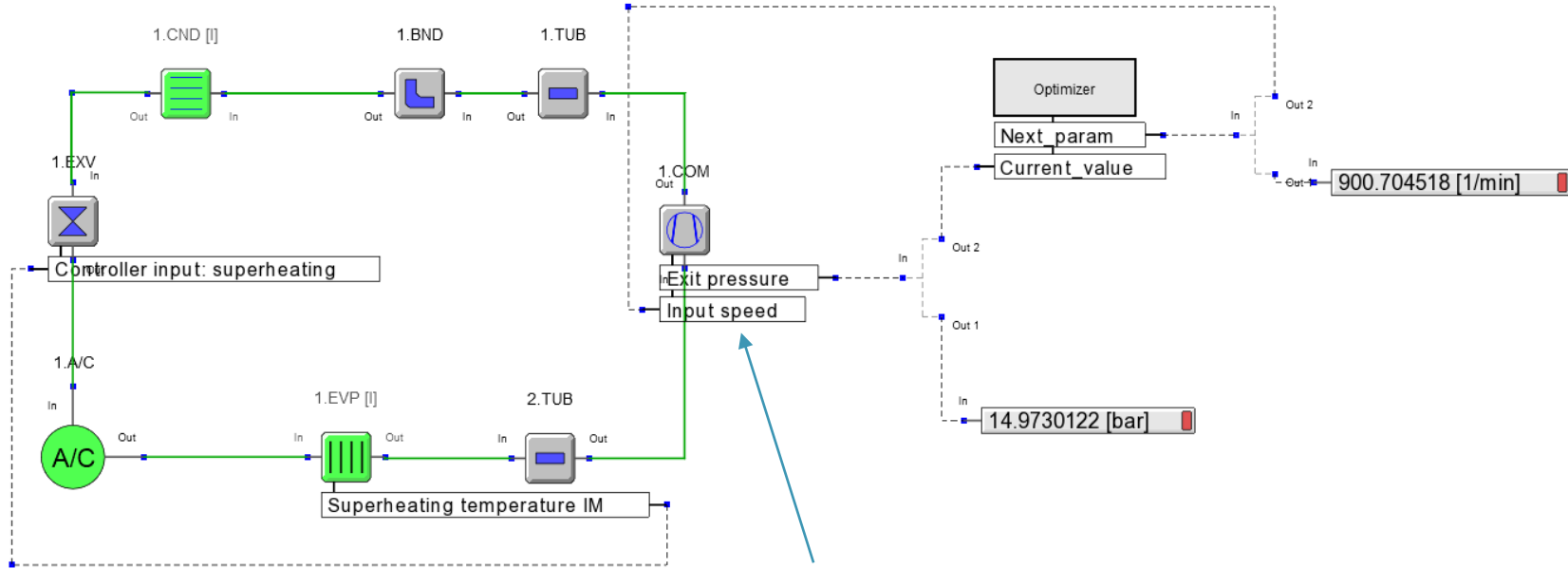
Transient Optimization

Motivation



- KULI has a powerful built-in optimization
 - Single or multiple targets
 - Optimization to value, minimum, maximum
 - Adjustable accuracy
 - Etc.
- For transient simulation optimization only works for single value obtained for complete time frame
 - E.g. integral of deviation between simulated and measured value
- Sometimes there is a demand to optimize a value for each timestep during a transient simulation

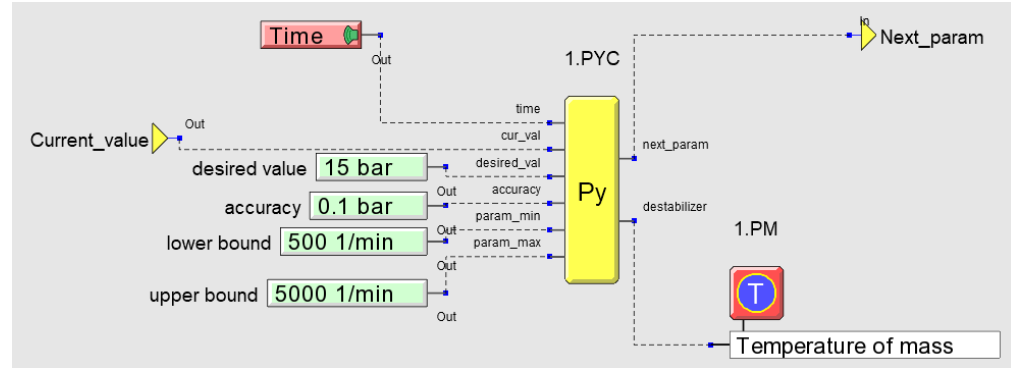
Transient Optimization of Compressor RPM



Adjust compressor rpm
such that desired discharge
pressure is achieved

Optimizer Subsystem

- Define bounds for parameter
 - Define desired value
 - Define desired accuracy
 - Read current time
 - Read current value
-
- Obtain next parameter value
 - Use value of “destabilizer” to prevent convergence until internal optimization has finished



Results and Details

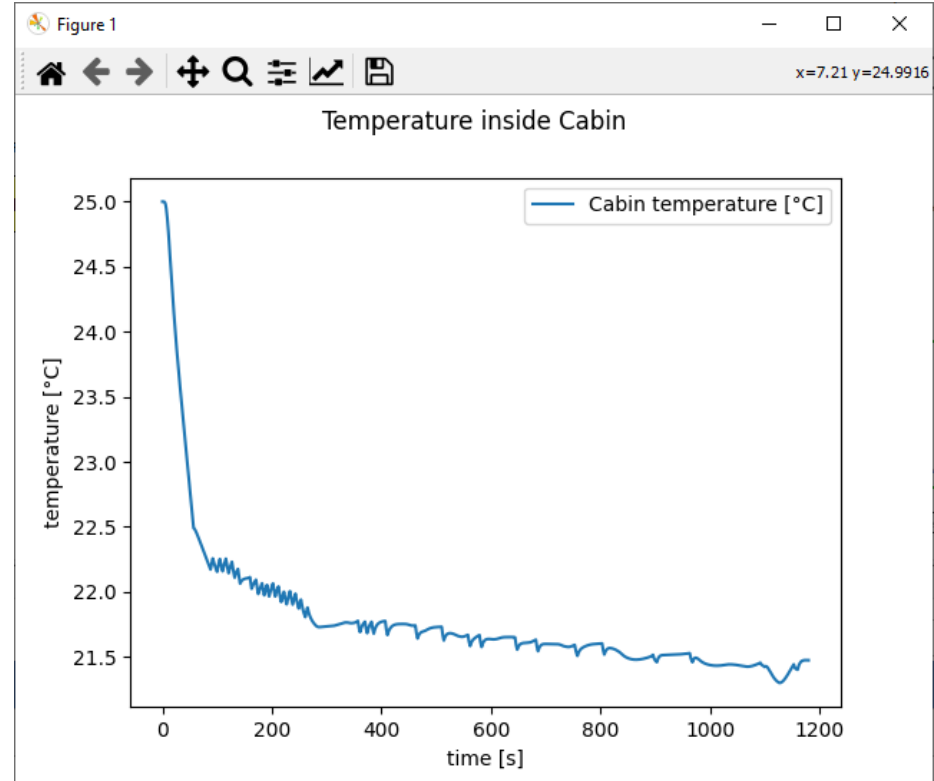
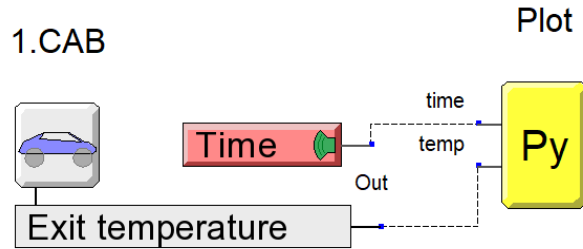
Ex_AC_trans_PYC.hdmb ▶			
Simulation parameters 1 ▶		AC Compressor ▶	
Time	Ambient temperature	Exit pressure	Input speed
s	°C	bar	1/min
0	32	14,9594	3191,407
1	32,8	14,9595	2635,4326
2	33,6	15,0272	2293,0227
3	34,4	14,9051	1947,8619
4	35,2	14,9288	1751,2092
5	36	14,9876	1597,9698
6	36,8	14,9756	1422,6583
7	37,6	14,934	1266,5924
8	38,4	14,9643	1146,9709
9	39,2	15,0014	1027,2338
10	40	14,973	900,7045

- Script implements a Newton method into the iteration process
- Method can also be used for steady state calculations
 - Separate function available
- Optional output of internal values possible

The background of the slide is a dark, abstract image. It features a complex network of blue and red lines that form wireframe-like structures, possibly representing architectural or engineering models. Interspersed among these are bright, glowing red and blue light trails that create a sense of motion and depth. The overall effect is high-tech and futuristic.

Plotting with Python

Create Figures During Runtime



Parameters for Plot Python Controller



- Function: `Scope1(t, x)`
for each time step of the simulation (input t), plot given values (input x)
- Initialization function:
`SetupFigure()`
before the start of the simulation, create an empty figure to where the data can be plotted.

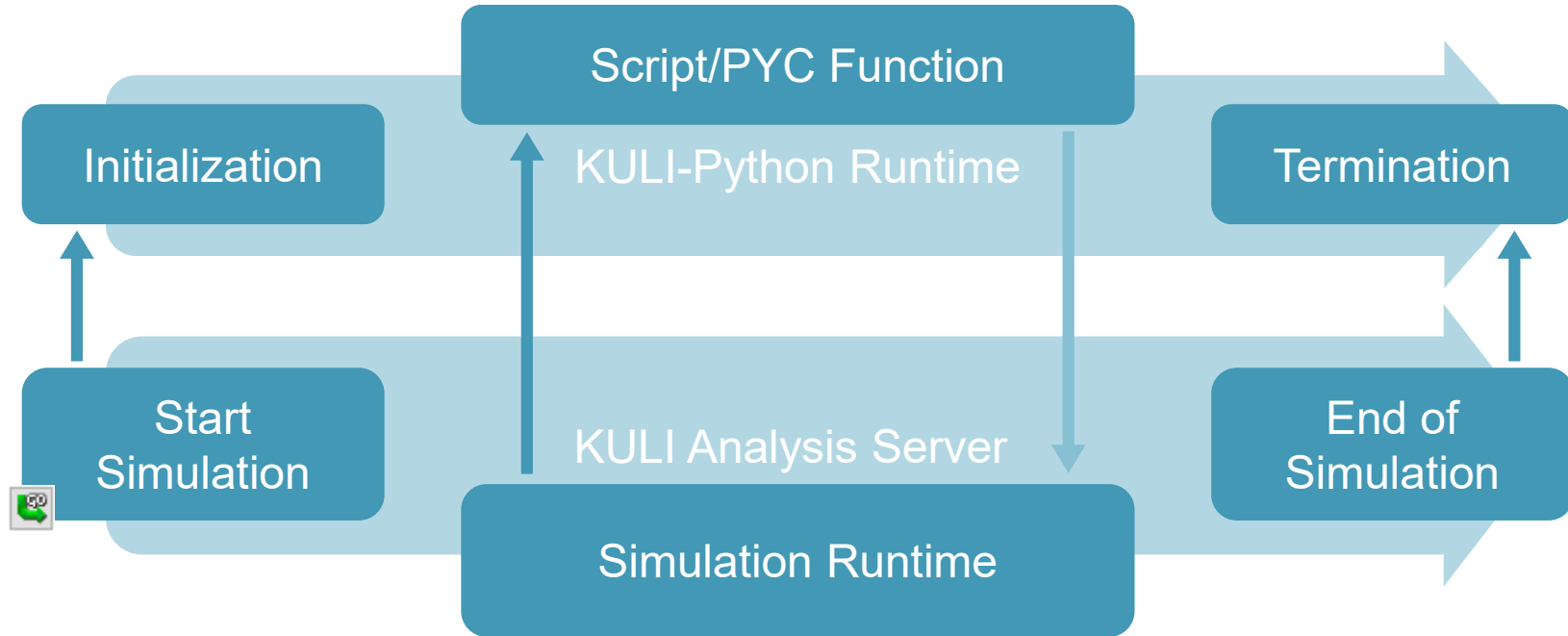
The screenshot shows the 'Component parameters' dialog box for a 'Python Controller'. The ID is '1.PYC' and the comment is 'Plot'. The Python module is 'C:\ECS\KULI_170000\data\PythonScripts\Scope1_keepOpen.py'. The function is 'Scope1(t, x)'. The initialization function is 'SetupFigure()'. The termination function is empty. The inputs table has two rows: 'time' with parameter 't' and unit 's', and 'temp' with parameter 'x' and unit '°C'. The outputs table is empty.

Identifier	Function parameter	Comment	Unit
time	t	simulation time	s
temp	x	temperature in passenger cabin	°C

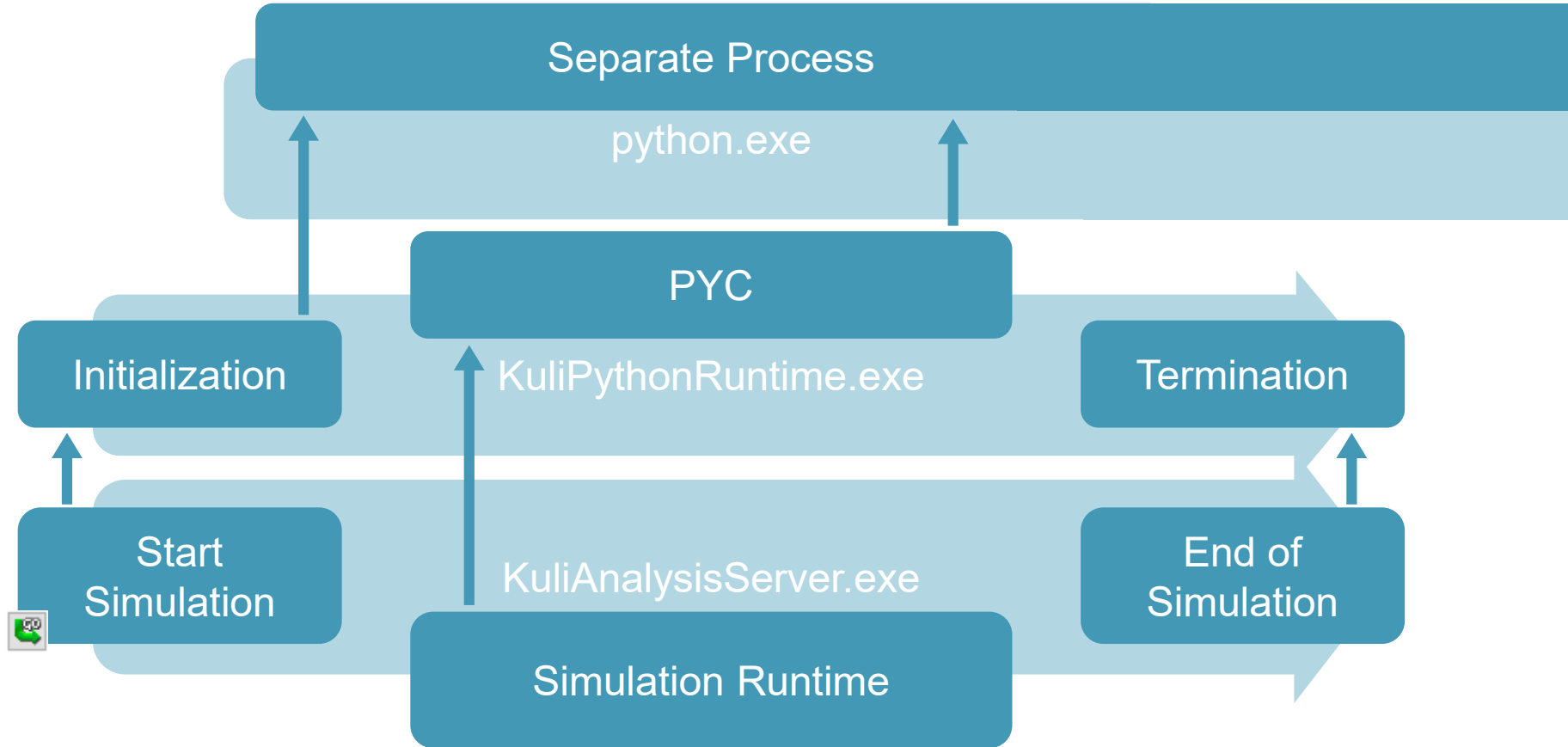
Identifier	Comment	Unit
------------	---------	------

De-Coupling KULI and Python

- How to keep a script running after end of simulation?



De-Coupling KULI and Python



Creating a Separate Process



- Define python.exe as executable for separate process
 - default: KuliPythonRuntime.exe, terminates with KuliAnalysisServer.exe
- Information is passed from `sconn` to `rconn` (send/receive)
- Separate Process executes function `DrawConsumer` and gets argument `rconn` (i.e. it only receives information)

initialization
function

```
import multiprocessing as mp
import sys
def SetupFigure():
    mp.set_executable(sys.exec_prefix + '/Python.exe')
    global sconn
    rconn, sconn = mp.Pipe(duplex=False)
    Process = mp.Process(target=DrawConsumer, args=(rconn,))
    Process.start()
```

PYC Function

- Is executed in each KULI iteration

```
def Scope1(t,x):  
    global sconn  
    sconn.send([x, t])
```

- access global variable `sconn` to send inputs x and t from KULI to the separate process
- Sends data from KULI to external process in each time step
- There, the plot is being created, updated and kept open

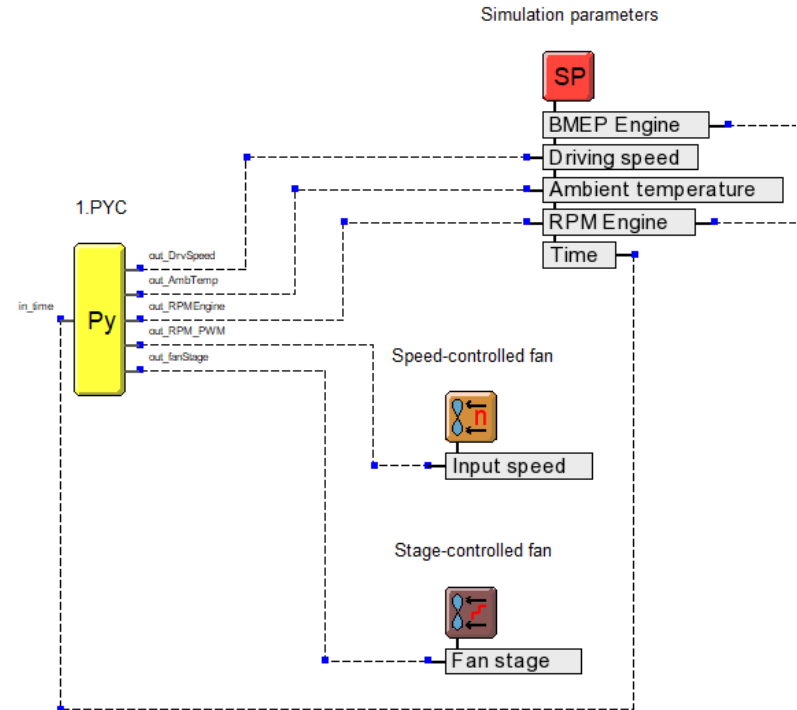
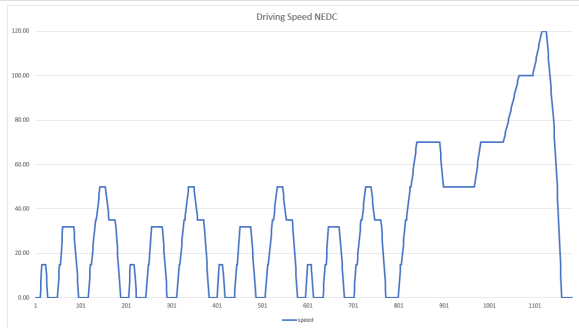
The background of the central banner is an abstract digital composition. It features a dense network of blue wireframe lines forming geometric shapes, overlaid with vibrant, glowing red light streaks that radiate from the center, creating a sense of dynamic energy and data flow.

Get operation points from Excel®

Given Data Set

- Excel® sheet containing some NEDC data

MS Excel®										
	A	B	C	D	E	F	G	H	I	J
1	Time [s]	RPMEngine [1/min]	BMEPEngine [bar]	DrvSpeed [km/h]	WarmUpTemp [K]	AmbTemp [°C]	Comment [text]	FanStage [integer]	Gear [integer]	RPM_pwm [1/min]
2	1	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
3	2	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
4	3	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
5	4	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
6	5	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
7	6	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
8	7	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
9	8	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
10	9	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
11	10	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
12	11	850	6.000000	0.000001	0.000000	40.000000	lorem ipsum	1	0	935
13	12	850	6.000000	3.750000	0.000000	40.000000	lorem ipsum	1	1	935
14	13	850	6.000000	7.500000	0.000000	40.000000	lorem ipsum	1	1	935
15	14	1323.3094	6.000000	11.750000	0.000000	40.000000	lorem ipsum	1	1	1455.640393
16	15	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
17	16	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
18	17	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
19	18	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
20	19	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
21	20	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
22	21	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
23	22	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
24	23	1689.3312	6.000000	15.000000	0.000000	40.000000	lorem ipsum	1	1	1858.264294
25	24	1407.776	6.000000	12.500000	0.000000	40.000000	lorem ipsum	1	1	1548.553601
26	25	1126.2208	6.000000	10.000000	0.000000	40.000000	lorem ipsum	1	1	1238.842908



The Python Controller

- **Function:** `getValuesExCar_PM(t)` for a time during the simulation (input t), read values from a given data source
- **Initialization function:** `readFromXLS()` before the start of the simulation, read all data from Excel® into the python process, such that it can be accessed during runtime



Component parameters

Python Controller ID: 1.PYC

Comment:

Python module: C:\ECS\KULI_170000\data\PythonScripts\getSimParam.py

Function: `getValuesExCar_Trans_PM(t)`

Initialization function: `readFromXLS()`

Termination function:

Inputs

Identifier	Function parameter	Comment	Unit
in_time	t	time	s

Outputs

Identifier	Comment	Unit
out_DrvSpeed		km/h
out_AmbTemp		°C
out_RPMEngine		1/min
out_RPM_PWM		1/min
out_fanStage		-

OK Cancel

The Python Script



```
Time = []
RPMEngine = []
BMEPEngine = []
DrvSpeed = []
WarmUpTemp = []
AmbTemp = []
Comment = []
FanStage = []
Gear = []
RPM_pwm = []

def readFromXLS():
    '''Initialization function: Imports simulation parameters from NEDC_SimParam.xlsx...

    file_name = "NEDC_SimParam.xlsx"

    excel_data = pd.read_excel(file_name) # read .xlsx file
    xlsheader = list(excel_data.columns.values) # get header of .xlsx file

    # get/set globals
    global Time
    Time = excel_data[xlsheader[0]].tolist()
    list(map(float, Time))
```

The Python Script



```
def getValuesExCar_Trans_PM(t):  
    '''For a given step in time, return the corresponding simulation parameters required for EX_Car_Trans_PM. ...  
  
    return interpValue(t, Time, DrvSpeed), \  
        interpValue(t, Time, AmbTemp), \  
        interpValue(t, Time, RPMEngine), \  
        interpValue(t, Time, RPM_pwm), \  
        deriveStage(t, Time, FanStage)
```

An abstract background image featuring a dark field with vibrant red and blue light trails streaking across it. In the background, there are faint, glowing wireframe structures that resemble architectural or technical designs.

Python-Simulink Interface

Co-Simulation with KULI as Master



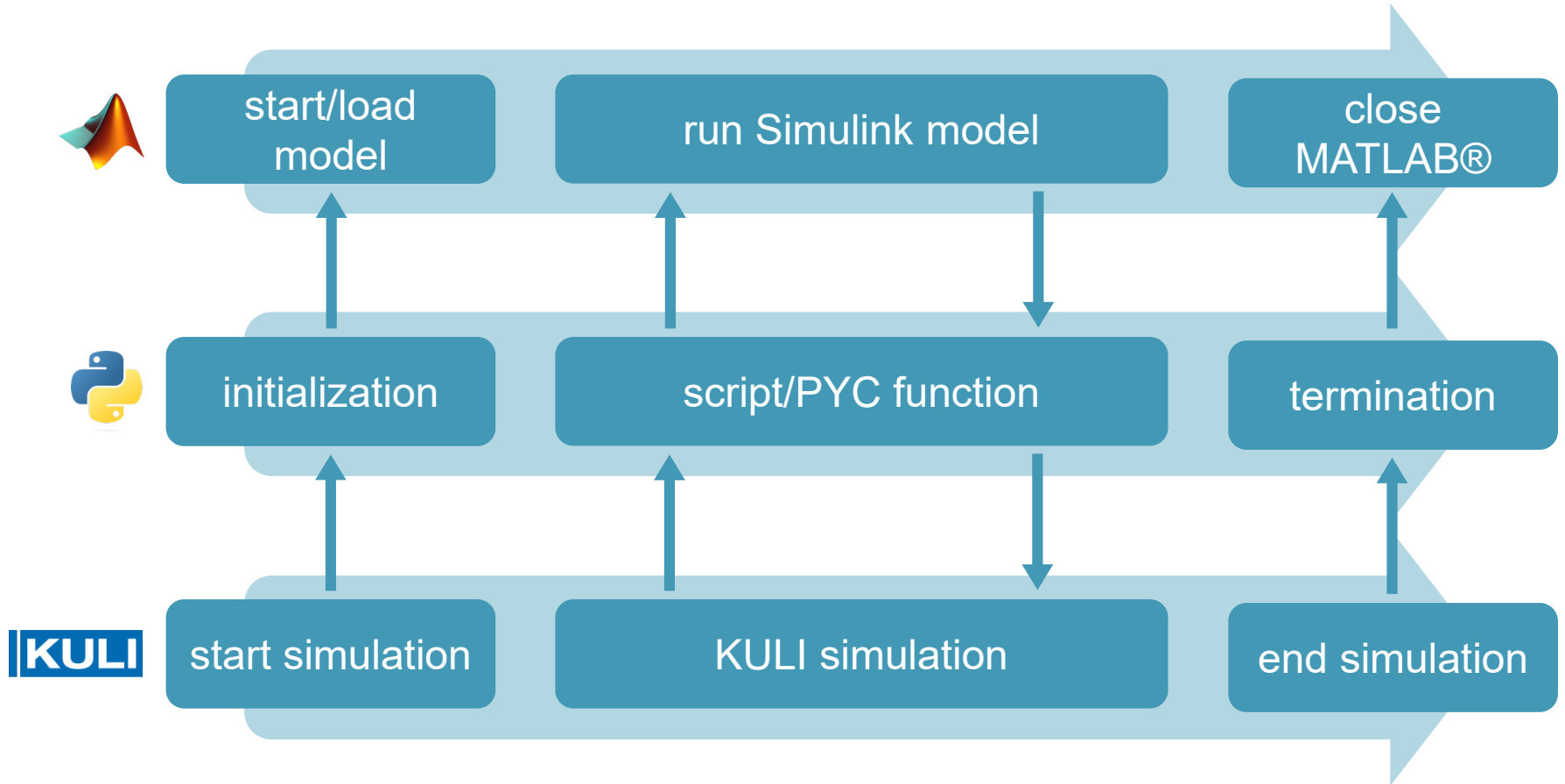
- Simulink is started from KULI at the start of simulation
- Both perform transient simulation
- They exchange data during simulation
- PYC eliminates disadvantages of built in Simulink Controller:
 - max number of inputs is restricted
 - only one output
 - strict formal criteria are required
 - only works with .mdl (not with .slx)

Assumption: Time Steps Match



- Co-simulation in general: simulation of two de-coupled models in parallel with communication
 - depend on each other with regard to the transmitted data
 - independent of each other with respect to the duration (start/end time) and the time step of both models
- Simplification:
start/end time and time step in both models are equal
 - of course it is also possible otherwise

Python as Interface



Interfaces

- KULI – Python: PYC
- Python – Simulink: “MATLAB® Engine API for Python”

```
import matlab.engine
```

- start MATLAB® and load Simulink models from Python
- send commands to MATLAB® from Python, e.g.

```
SEngine.get_param(sl_model, 'SimulationStatus')
```

The Python Script



- Inputs:
 - current time
 - any number of inputs necessary for Simulink model
- Use global variables
 - `Lasttime = -1` to track time steps
 - `prev_result = 0` to „remember“ result(s) from previous iteration
 - define a „prev“ global for each desired output
- Once during each time step, data is exchanged

The Python Script



- Initialization function
 1. Load specified MATLAB® Simulink model
 2. Start the model and instantly pause it
- Main function
 1. IF new time step, THEN
 1. set inputs for Simulink from KULI
 2. perform a step in time in Simulink
 3. return the outputs from Simulink to KULI
 2. ELSE
 1. return previous result
- Termination function
 1. Quit MATLAB® Simulink

Co-Simulation with other Software



- Co-simulation is possible for every software that offers a Python (or COM) interface
- FMI: FMPPy (has already been implemented)
- Motor-CAD: PyMotorCAD
- Idea for co-simulation is the same regardless of the software

The background of the slide is a horizontal band with a dark, abstract design. It features glowing red and blue lines that form wireframe-like structures, possibly representing architectural or engineering models. The red lines are more prominent in the center, while blue lines are on the sides.

Conclusion

Conclusion



- Extend functionality of KULI
 - Custom functions (e.g. data analysis)
- Make use of Python modules (libraries)
 - `matplotlib`
 - `numpy`
 - `CoolProp`
 - many more
- Create interface for 3rd party software applications

Visit our Online Library!



- Access via online search function

The screenshot displays the KULI software interface. On the left, a 'General data' panel shows user information: User: ECS, Date: Montag, 10. November 2014 10:52:01, Department: , Purpose: Cooling system with a hydraulic circui, Results: , Report No.: , and Memo: . The main area shows a table of components with columns for ID, Width Diameter, and Height [mm].

ID	Width Diameter	Height [mm]
- Components		
1.CP value	-	-
1.Inlet Grid	500	300
1.Stage-controlled fan	290	290
1.Built-in resistance	-	-
1.Area resistance	620	250
1.Charge air cooler	620	300
1.Speed-controlled fan	300	300
1.Par.Flow cooler	100	100
1.Radiator	600	400
- Circuits / Air Path		

On the right, a 'python' window is open, displaying a help document with the following sections:

- Python-Simulink Interface**
OverviewThe block diagram tool MATLAB® Simulink is widely used for the creation and simulation of physical models, particularly in the automotive industry. KULI advanced provides a feature called the Simulink Controller which allows the user to run MATLAB® simulations from the KULI user interface. However, this Simulink Controller is subject to several restrictio...
- Plotting with Python Controller**
OverviewThe Python controller component included in the KULI advanced module provides the possibility to visualize any quantities in a plot during the runtime of a simulation. This example provides a Python script that gives the user access to multiple plotting methods for KULI simulations. The example is based on theTutorial Example - Transient Simulation of a C...
- Python Interface**
With the help of Python controllers KULI gives the possibility to include Python Scripts into KULI simulation models.NoteThe KULI installer does not install Python itself. Currently the Python versions 3.7 to 3.11 are supported. One of the mentioned Python versions has to be installed, manually, prior to starting KULI.

- Detailed examples
- Out-of-the-box solutions



Forward. For all.