Simulation programming manual

This manual gives guidelines for creating simulations for SFDEsim simulator. Simulations are programmed as python modules which must follow certain principles and be constructed from certain objects. Outside of these restrictions the simulations can be programmed freely. In addition to this manual, already existing simulations can be used as examples and a template for simulation is provided in the simulation files directory.

When simulation is created, it has to be included in the Simulation\_list.json file in the PythonModules\Simulation\_files directory. In this file a “subjects” dictionary contains dictionaries for all courses. The new simulation can be added to existing course or new course dictionary can be created. The course is used in the main menu to organize the simulations and has no other effect. In the course dictionary, new simulation dictionary has to be created for the new simulation. This dictionary must have 2 keys containing strings. The “filename” string must contain the filename of the simulation (file extension included) and the “abstract” string is for freely written description for the simulation which is displayed in the main menu. The maximum length of one row in the abstract is approximately 90 characters (on 1080x1920 monitors), therefore using line break (\n) to make longer description to wrap on multiple rows is recommended.

A PDF-manual for simulation can be created and opened from the simulator. The contents of the manual are up to the simulation programmer. In order to enable the simulation to open from the simulator program’s help-menu, the manual has to be located in the (simulator’s) Documents folder and named as simulation name + “\_manual”.

Imports

At the top of the simulation file, all required modules must be imported. The required modules are listed in the table below (table 1). The simulator includes custom modules for graphing, other graphical widgets and additional mathematical functions which can be imported from given modules in table 1. For this the pathing has to be set correctly, example can be seen in the template or any other simulation. Other python modules can be imported freely as well if required in your simulation.

**Table 1.** Simulation module imports.

|  |  |  |
| --- | --- | --- |
| Module name | Attributes to be imported | Required |
| sys | All | X |
| os | All | X |
| PySide6.QtCore | QObject, Slot | X |
| PySide6.QtWidgets | QWidget, QGridLayout (or any other layout) | X |
| Numpy | array (import as np, or change all use instaces) | X |
| PhasorPlotWidget | PhasorGraphWidget |  |
| LinePlotWidget | LinePlotWidget |  |
| SimulationWindowWidgets | ParameterViewWidget, PictureViewWidget |  |
| SimuMath | cart2pol, angle\_loop\_rad, pol2cart, solve\_2bus\_NR, solve\_power\_flow\_GS, reference frame transformations |  |

parameters class

The parameters class is used to declare high level parameters for simulation. In the \_\_init\_\_ method of the class, simulation name, time step, simulation start time, graphing interval and the user editable input parameters are set. The editable parameters are set in the form of nested dictionary and includes the setup for both the numeric and dropdown parameters. The structure of the parameters is shown in table below (table 2). The SI unit prefixes are shown as a dropdown menu for a parameter and can be set from pico- (10^-12) to tera- (10^12). Only the main prefixes at powers of three are supported. If no prefixes are used can both limits be set to zero. The graphing\_interval describes how often the graphical elements are updated as this is done only ones every x simulation steps (x representing the graphing\_interval), in order to reduce computational load of graphing. More information in the user manual

**Table 2.** Attributes in parameters class.

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | | | Type |
| self.simulation\_name | | | str |
| self.steptime | | | float |
| self.simulation\_time | | | float |
| self.graphing\_interval | | | int |
| self.input\_parameters | | | dict |
|  | numeric parameter key | | dict key |
|  |  | display\_name | str |
|  |  | symbol | str |
|  |  | editable | bool |
|  |  | type | str |
|  |  | init\_value | float |
|  |  | unit | str |
|  |  | default\_prefix | int |
|  |  | prefix\_limits | list |
|  |  | maximum | float |
|  |  | minimum | float |
|  | dropdown parameter key | | dict key |
|  |  | display\_name | str |
|  |  | symbol | str |
|  |  | editable | bool |
|  |  | type | str |
|  |  | items | list |
|  |  | unit | str |

simulator class

The simulator class contains the main simulation part of the simulation module. The class inherits QObject, which enables the simulation code to be executed in its own thread. This both increases the performance of the simulation but also keeps the user interface responsive during the simulation run. The class has three methods \_\_init\_\_, update\_matrixes and run. The \_\_init\_\_ method initializes the simulation and should be used for setting up simulations interfacing to lowers the program, declaring the input variables and doing all pre-simulation calculations. The reguired initializations are shown in the template. Self.input\_variables and self.input\_texts are numpy array containing all the numeric input variables and list containing all dropdown variables respectively. Additionally, functionality for setting prefixes correctly is included in the template. At the end of the method self.update\_matrixes must be called. Other functionality can be added before the self.update\_matrixes call.

The update matrixes method is called at the end of the \_\_init\_\_ method of the simulator class and when the user changes input variables or presses the “update” button depending on the “autoupdate” settings. In this method, the calculations that can be only done when inputs are changes can be done in this method in order to reduce the calculation load during the simulation step calculation.

The run method is executed for each simulation step and should include the simulation step calculations. The method operates as a “Slot” with Boolean input and requires a corresponding decorator to be used (see template). This method includes a while loop with condition of self.flow\_control methods output. And ends with call of self.send\_to\_graph method. This method call takes a list containing all variables for graphical widgets as an input. All simulation calculations should be written in the while loop. Adding other methods to the class can be done if needed for creation of the simulation.

graphicsViewWidget class

The graphicsViewWidget inherits QWidget and is placed as the central widget into the window. In this object all the graphics elements are created and placed into. Four custom widgets and all built in PyQt widgets can be used to build graphics of the simulation. LinePlotWidget can be used to create time domain graphs, PhasorViewWidget can be used to create phasor domain graphs, ParameterViewWidget can be used to create easily updateable text-based parameter outputs and PictureViewWidget can be used to add pictures to the simulation view. Examples of all these custom widgets can be found in the template. The building of the simulation view is done in the \_\_init\_\_ method. When the sub-widgets are created, they have to placed be into the graphicsViewWidget using a layout. In the template QGridLayout is used, which enables the sub-widgets to be placed in a grid formation.

The updating of the graphicsViewWidget is done in the update method, which is executed for each graphing interval. The method operates as a “Slot” with list input and requires a corresponding decorator to be used (see template). The list given in the send\_to\_graph method call in the run method of the simulation class is received here as an input. In the template the input is named as “inp”, which can be changed. The updating methods for the custom widgets are explained below. The last line of the update method must be the call for the graphing\_flow\_control, which allows the simulation mathematics to stay synchronized with the simulation graphics.

LinePlotWidget

LinePlotWidget is a custom widget for time domain plotting based on PyQtGraph. The custom features enable easier updating, custom markers, line settings changes and control buttons. The control buttons include zoom in and out buttons for x-axis, and buttons for toggling each of the plotlines visible/hidden. When an instance of the widget is created it takes a graph name, x-axis title, y-axis title number of x-axis data points, simulation time step, graphing interval and legend on/off setting. The plots are added to the graph and given a name, which is used when referring to the plot while changing its settings or updating it. The LinePlotWidget methods for simulator programmer explained in the table below (table 3). As the widget is based on the PyQtGraph module, some of the features of the module are included, for example the graph right click options. More about this in the SFDEsim simulator user manual and on the pyqtgraph.org website.

**Table 3.** LinePlotWidget methods for simulation programmer.

|  |  |
| --- | --- |
| Method | Description |
| add\_plotline | Adds a new plot to the graph. |
| Takes a plot name, x-axis data, y-axis data and line color. |
| The axis data type is numpy array. If the given arrays are not as long as the x length of the graph, the rest of the values are substituted as zeros. |
|
| update | Updates the given plots data. |
| Takes plot name and new x and y-axis data as inputs. |
| step | Appends a new data point to a plot and removes the oldest data point. |
| Takes plot name and new y axis value as inputs |
| set\_text | Changes the graph texts. |
| Takes graph title, text, color and font size, and x and y-axis label texts color and font size as inputs. |
| Only updates the texts of which the text is changed. |
| change\_line\_pen | Changes the plotline line type. |
| For a given plot, the line color, width and linetype are taken as inputs. |
| Colors can be given as color names (e.g. red, green, cyan), rgb values or hexadecimal color codes. |
| Line type can be set to continuous (-), dashed (--) or dotted (..) |
| Line width is given as decimal number. |
| Only the given parameters are changed, no need to update all every time. |

PhasorPlotWidget

PhasorPlotWidget is a custom widget for phasor domain plotting based on PyQtGraph. The custom features allow simple creation and update of phasors and a polar coordinate background. The phasors are added to the graph and given a name, which is used when referring to the phasor while changing its settings or updating it. The PhasorPlotWidget methods for simulator programmer explained in the table below (table 4). As the widget is based on the PyQtGraph module, some of the features of the module are included, for example the graph right click options. More about this in the SFDEsim simulator user manual and on the pyqtgraph.org website.

**Table 4.** PhasorPlotWidget methods for simulation programmer.

|  |  |
| --- | --- |
| Method | Description |
| add\_phasor | Adds a new phasor to the graph. |
| Takes phasor name, x and y values for the start coordinate of the phasor, x and y values for the end coordinate of the phasor and the phasor color. |
| update | Updates a given phasor to new coordinates. |
| Takes phasor name, new x and y values for the start coordinate of the phasor, and new x and y values for the end coordinate of the phasor. |
| change\_phasor\_pen | Changes the phasor line type. |
| For a given phasor, the line color, width and linetype are taken as inputs. |
| Colors can be given as color names (e.g. red, green, cyan), rgb values or hexadecimal color codes. |
| Phasor line type can be set to continuous (-), dashed (--) or dotted (..) |
| Phasor width is given as decimal number. |
| Only the given parameters are changed, no need to update all every time. |

ParameterViewWidget

ParameterViewWidget is a custom widget for showing simulation parameters of output values in a numeric format. The advantages of using ParameterViewWidget over a basic QLabels are the uniform formatting for multiple parameters, simplified updating, data formatting options and automatic unit prefix updating. When creating instance of the widget a header and rounding decimal are given. After this a number of rows can be added with each getting a parameter name, initial value and unit. Each row is referred to with its index, where the first line added will have an index of 0. The widget supports three numeric formats for the value; scalar and complex number in rectangular and polar forms. Between rows a separator line can be added using the add\_line method. The separator line is placed after the previously initialized row but does not take up a row index. Update\_row, update\_complex\_ row and update\_polar\_row methods can be used to update the values of a row in the wanted format. The update method changes the value of a row to the given new value rounded to the widget’s rounding precision and adjusts the value and the unit prefix to show the value in a concise form. A change value method can be used to only change the value without updating methods’ features.

PictureViewWidget

PictureViewWidget is used for placing a picture into the simulation view. When creating a instance of the widget a picture name, header and directory are given as inputs. Picture name is the file name of the picture including the file extension. Header is optional an is placed above the picture is given, And the directory of the picture can be given if special directory is used. By default the picture location is /Simulation\_files/Pictures.