16 OutStream system

The PRA and UQ framework provides the capabilities to visualize and dump out the data that are generated, imported (from a system code) and post-processed during the analysis. These capabilities are contained in the "OutStream" system. Actually, two different OutStream types are available:

- Print, module that lets the user dump the data contained in the internal objects;
- Plot, module, based on MatPlotLib [4], aimed to provide advanced plotting capabilities.

Both the types listed above accept as "input" a DataObjects object type. This choice is due to the "DataObjects" system (see section 14) having the main advantage of ensuring a standardized approach for exchanging the data/meta-data among the different framework entities. Every module can project its outcomes into a DataObjects object. This provides the user with the capability to visualize/dump all the modules' results. Additionally, the Print system can accept a ROM and inquire some of its specialized methods. As already mentioned, the RAVEN framework input is based on the eXtensible Markup Language (XML) format. Thus, in order to activate the "OutStream" system, the input needs to contain a block identified by the `<OutStreams>` tag (as shown below).

```
<OutStreams>
  <!-- "OutStream" objects that need to be created-->
</OutStreams>
```

In the "OutStreams" block an unlimited number of "Plot" and "Print" sub-blocks can be specified. The input specifications and the main capabilities for both types are reported in the following sections.

16.1 Printing system

The Printing system has been created in order to let the user dump the data, contained in the internal data objects (see Section 14), out at anytime during the calculation. Additionally, the user can inquire special methods of a ROM after training it, through a printing step. Currently, the only available output is a Comma Separated Value (CSV) file for DataObjects, and XML for ROM objects. This will facilitate the exchanging of results and provide the possibility to dump the solution of an analysis and "restart" another one constructing a data object from scratch, as well as access advanced features of particular reduced order models.

16.1.1 DataObjects Printing

The XML code, that is reported below, shows different ways to request a Print OutStream for DataObjects. The user needs to provide a name for each sub-block (XML attribute). These names are then used in the Step blocks to activate the Printing keywords at any time. The XML node has the following available attributes:

- `name`, required string attribute, is a user-defined identifier for this data object. Note: As with other objects, this name can be used to refer to this specific entity from other input blocks in the XML.

As shown in the examples below, every `<Print>` block must contain, at least, the two required tags:

- `<type>`, the output file type (csv or xml). Note: Only csv is currently available for `<DataObjects>`
- `<source>`, the Data name (one of the Data items defined in the `<DataObjects>` block.

An optional tag `<filename>` can be used to specify the filename for the output. If this is not defined, then the default name will be the `name` identifier of the tag.

If only these two tags are provided (as in the "first-example" below), the output file will be filled with the whole content of the "d-name" Data object.

```
<OutStreams>
  <Print name='first-example'>
    <type>csv</type>
    <source>d-name</source>
  </Print>
  <Print name='second-example'>
    <type>csv</type>
    <source>d-name</source>
    <what>Output</what>
  </Print>
  <Print name='third-example'>
    <type>csv</type>
    <source>d-name</source>
    <what>Input</what>
  </Print>
  <Print name='fourth-example'>
    <type>csv</type>
    <source>d-name</source>
    <what>Input|var-name-in,Output|var-name-out</what>
```

```
  </Print>
  <Print name='fifth-example'>
    <type>csv</type>
    <source>d-name</source>
    <filename>example5</filename>
  </Print>
</OutStreams>
```

If just part of the <source> is important for a particular analysis, the additional XML tag <what> can be provided. In this block, the variables that need to be dumped must be specified, in comma separated format. The available options, for the <what> sub-block, are listed below:

- Output, the output space will be dumped out (see "second-example")
- Input, the input space will be dumped out (see "third-example")
- Input—var-name-in/Output—var-name-out, only the particular variables "var-name-in" and "var-name-out" will be reported in the output file (see "fourth-example")

Note all of the XML tags are case-sensitive but not their content.

16.1.2 ROM Printing

While all ROMs in RAVEN are designed to be used as surrogate models, some ROMs additionally offer information about the original model that isn't accessible through another means. For instance, HDMRRom objects can calculate sensitivity coefficients for subsets of the input domain. The XML code shown below demonstrates the methods to request these features from a ROM. The user needs to provide a <name> for each sub-block (XML attribute). These names are then used in the Step blocks to activate the Printing keywords at any time. As shown in the examples below, every <Print> block for ROMs must contain, at least, the three required tags

- <type>, the output file type (csv or xml). Note: Only xml is currently available for ROMs
- <source>, the ROM name (one of the <ROM> items defined in the <Models> block.
- <what>, the comma-separated list of desired metrics. The list of metrics available in each ROM is listed under that ROM type in Section 17.3. Alternatively, the keyword 'all' can be provided to request all available metrics, if any.

Addtionally, when printing ROMs one optional node is available,

- <target>, the ROM target for which to inquire data

If the ROM is time-dependent, the printed properties will be collected by time step. ROM printing uses the same naming conventions as DataObjects printing. Examples:

```
<OutStreams>
  <Print name='first-ROM-example'>
    <type>xml</type>
    <source>mySobolRom</source>
    <what>all</what>
  </Print>
  <Print name='second-ROM-example'>
    <type>xml</type>
    <source>myGaussPolyRom</source>
    <what>mean,variance</what>
  </Print>
</OutStreams>
```

16.2 Plotting system

The Plotting system provides all the capabilities to visualize the analysis outcomes, in real-time or as a post-processing stage. The system is based on the Python library MatPlotLib [4]. MatPlotLib is a 2D/3D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. This external tool has been wrapped in the RAVEN framework, and is available to the user. Since it was unfeasible to support, in the source code, all the interfaces for all the available plot types, the RAVEN Plotting system directly provide a formatted input structure for 11 different plot types (2D/3D). The user may request a plot not present among the supported ones, since the RAVEN Plotting system has the capability to construct on the fly the interface for a Plot, based on XML instructions.

16.2.1 Plot input structure

In order to create a plot, the user needs to add, within the `<OutStreams>` block, a `<Plot>` sub-block. Similar to the `<Print>` OutStream, the user needs to specify a `name` as an attribute of the plot. This name will then be used to request the plot in the `<Steps>` block. In addition, the Plot block accepts the following attributes:

- `interactive`, optional bool attribute, specifies if the Plot needs to be interactively created (real-time screen visualization).
  Default: False
- `overwrite`, optional bool attribute, used when the plot needs to be dumped into picture file/s. This attribute determines whether the code needs to overwrite the image files every time a new plot (with the same name) is requested.
  Default: False

An optional tag <filename> can be used to specify the filename for the output. If this is not defined, then the default base name will be the name identifier of the tag prepended and appended with extra information that identifies the plot further.

As shown, in the XML input example below, the body of the Plot XML input contains two main sub-nodes:

- <actions>, where general control options for the figure layout are defined (see Section 16.2.1.1).
- <plotSettings>, where the actual plot options are provided.

These two main sub-block are discussed in the following paragraphs.

16.2.1.1 "Actions" input block

The input in the <actions> sub-node is common to all the Plot types, since, in it, the user specifies all the controls that need to be applied to the figure style. This block must be unique in the definition of the <Plot> main block. In the following list, all the predefined "actions" are reported:

- <how>, comma separated list of output types:
  - screen, show the figure on the screen in interactive mode
  - pdf, save the figure as a Portable Document Format file (PDF). Note: The pdf format does not support multiple layers that lay on the same pixel. If the user gets an error about this, he/she should move to another format.
  - png, save the figure as a Portable Network Graphics file (PNG)
  - eps, save the figure as an Encapsulated Postscript file (EPS)
  - pgf, save the figure as a LaTeX PGF Figure file (PGF)
  - ps, save the figure as a Postscript file (PS)
  - gif, save the figure as a Graphics Interchange Format (GIF)
  - svg, save the figure as a Scalable Vector Graphics file (SVG)
  - jpeg, save the figure as a jpeg file (JPEG)
  - raw, save the figure as a Raw RGBA bitmap file (RAW)
  - bmp, save the figure as a Windows bitmap file (BMP)
  - tiff, save the figure as a Tagged Image Format file (TIFF)
  - svgz, save the figure as a Scalable Vector Graphics file (SVGZ)
- <title>, as the name suggests, within this block the user can specify the title of the figure. In the body of this node, a few other tags are available:

  - <text>, string type, title of the figure
  - <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

    The kwargs block is able to convert whatever string into a python type (for example <param1>{'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67]</param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.title" method in [4].

- <labelFormat>, within this block the default scale formatting can be modified. In the body, a few other tags are available:
  - <axis>, string, the axis where to apply the defined format, 'x,' 'y,' or 'both'.
    Default: 'both' Note: If this action will be used in a 3-D plot, the user can input 'z' as well and 'both' will apply this format to all three axis.
  - <style>, string, the style of the number notation, 'sci' or 'scientific' for scientific, 'plain' for plain notation.
    Default: scientific
  - <scilimits>, tuple, (m, n), pair of integers, if style is 'sci', scientific notation will be used for numbers outside the range 10^m to 10^n. Use (0,0) to include all numbers. Note: The value for this keyword, needs to be specified between brackets [for example, (5,6)].
    Default: (0,0)
  - <useOffset>, bool or double, if True, the offset will be calculated as needed; if False, no offset will be used; if a numeric offset is specified, it will be used.
    Default: False

- <figureProperties>, within this block the user specifies how to customize the figure style/quality. Thus, through this "action" the user has got full control on the quality of the figure, its dimensions, etc. This control is performed by the following keywords:
  - <figsize>, tuple (optional), (width, hight), in inches.
  - <dpi>, integer, dots per inch.
  - <facecolor>, string, set the figure background color (please refer to "matplotlib.figure.Figure" in [4] for a list of all the colors available).
  - <edgecolor>, string, the figure edge background color (please refer to "matplotlib.figure.Figure" in [4] for a list of all the colors available).

  - <linewidth>, float, the width of lines drawn on the plot.
  - <frameon>, bool, if False, suppress drawing the figure frame.

- <range>, the range "action" specifies the ranges of all the axis. All the keywords in the body of this block are optional:
  - <ymin>, double (optional), lower boundary for the y axis.
  - <ymax>, double (optional), upper boundary for the y axis.
  - <xmin>, double (optional), lower boundary for the x axis.
  - <xmax>, double (optional), upper boundary for the x axis.
  - <zmin>, double (optional), lower boundary for the z axis. Note: This keyword is effective in 3-D plots only.
  - <zmax>, double (optional), upper boundary for the z axis. Note: This keyword is effective in 3-D plots only.

- <camera>, the camera item is available in 3-D plots only. Through this "action," it is possible to orientate the plot as one wishes. The controls are:
  - <elevation>, double (optional), stores the elevation angle in the z plane.
  - <azimuth>, double (optional), stores the azimuth angle in the x,y plane.

- <scale>, the scale block allows the specification of the axis scales:
  - <xscale>, string (optional), scale of the x axis. Three options are available: "linear,""log," or "symlog."
    Default: linear
  - <yscale>, string (optional), scale of the y axis. Three options are available: "linear," "log," or "symlog."
    Default: linear
  - <zscale>, string (optional), scale of the z axis. Three options are available: "linear," "log," or "symlog."
    Default: linear Note: This keyword is effective in 3-D plots only.

- <addText>, same as title.
- <autoscale>, is a convenience method for simple axis view autoscaling. It turns autoscaling on or off, and then, if autoscaling for either axis is on, it performs the autoscaling on the specified axis or axes. The following sub-nodes may be specified:
  - <enable>, bool (optional), True turns autoscaling on, False turns it off. None leaves the autoscaling state unchanged.
    Default: True

- <axis>, string (optional), determines which axis to apply the defined format, 'x,' 'y,' or 'both.'
  Default: 'both' Note: If this action is used in a 3-D plot, the user can input 'z' as well and 'both' will apply this format to all three axis.
- <tight>, bool (optional), if True, sets the view limits to the data limits; if False, let the locator and margins expand the view limits; if None, use tight scaling if the only output is an image file, otherwise treat tight as False.

- <horizontalLine>, this "action" provides the ability to draw a horizontal line in the current figure. This capability might be useful, for example, if the user wants to highlight a trigger function of a variable. The following sub-nodes may be specified:
  - <y>, double (optional), sets the y-value for the line.
    Default: 0
  - <xmin>, double (optional), is the starting coordinate on the x axis.
    Default: 0
  - <xmax>, double (optional), is the ending coordinate on the x axis.
    Default: 1
  - <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

    The kwargs block is able to convert whatever string into a python type (for example <param1>{'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67]</param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.axhline" method in [4].

  Note: This capability is not available for 3-D plots.

- <verticalLine>, similar to the "horizontalLine" action, this block provides the ability to draw a vertical line in the current figure. This capability might be useful, for example, if the user wants to highlight a trigger function of a variable. The following sub-nodes may be specified:
  - <x>, double (optional), sets the x coordinate of the line.
    Default: 0
  - <ymin>, double (optional), starting coordinate on the y axis.
    Default: 0
  - <ymax>, double (optional), ending coordinate on the y axis.
    Default: 1

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

The kwargs block is able to convert whatever string into a python type (for example <param1>{'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67]</param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.axvline" method in [4].

Note: This capability is not available for 3-D plots.

- <horizontalRectangle>, this "action" provides the ability to draw, in the current figure, a horizontally orientated rectangle. This capability might be useful, for example, if the user wants to highlight a zone in the plot. The following sub-nodes may be specified:
  - <ymin>, double (required), starting coordinate on the y axis.
  - <ymax>, double (required), ending coordinate on the y axis.
  - <xmin>, double (optional), starting coordinate on the x axis. Default: 0
  - <xmax>, double (optional), ending coordinate on the x axis. Default = 1
  - <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

The kwargs block is able to convert whatever string into a python type (for example <param1>{'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67]</param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.axhspan" method in [4].

Note: This capability is not available for 3D plots.

- <verticalRectangle>, this "action" provides the possibility to draw, in the current figure, a vertically orientated rectangle. This capability might be useful, for example, if the user wants to highlight a zone in the plot. The following sub-nodes may be specified:
  - <xmin>, double (required), starting coordinate on the x axis.
  - <xmax>, double (required), ending coordinate on the x axis.

- <ymin>, double (optional), starting coordinate on the y axis.
  Default: 0
- <ymax>, double (optional), ending coordinate on the y axis.
  Default: 1
- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1>{'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67]</param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.axvspan" method in [4].

Note: This capability is not available for 3D plots.

- <axesBox>, this keyword controls the axes' box. Its value can be 'on' or 'off'.
- <axisProperties>, this block is used to set axis properties. There are no fixed keywords. If only a single property needs to be set, it can be specified as the body of this block, otherwise a dictionary-like string needs to be provided. For reference regarding the available keys, refer to "matplotlib.pyplot.axis" method in [4].
- <grid>, this block is used to define a grid that needs to be added in the plot. The following keywords can be inputted:
  - <b>, boolean (required), toggles the grid lines on or off.
  - <which>, double (required), ending coordinate on the x axis.
  - <axis>, double (optional), starting coordinate on the y axis.
    Default: 0
  - <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

    The kwargs block is able to convert whatever string into a python type (for example <param1>{'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67]</param2> into a list, etc.).

16.2.1.2 "plotSettings" input block

The sub-block identified by the keyword `<plotSettings>` is used to define the plot characteristics. Within this sub-section at least a `<plot>` block must be present. the `<plot>` sub-section may not be unique within the `<plotSettings>` definition; the number of `<plot>` sub-blocks is equal to the number of plots that need to be placed in the same figure.

If sub-plots are to be defined then `<gridSpace>` needs to be present. `<gridSpace>` specifies the geometry of the grid that a subplot will be placed. The number of rows and number of columns of the grid need to be set.

For example, in the following XML cut, a "line" and a "scatter" type are combined in the same figure.

```
<OutStreams>
  <Plot name='example2PlotsCombined'>
    <actions>
      <!-- Actions -->
    </actions>
    <plotSettings>
      <gridSpace>2 2</gridSpace>
      <plot>
        <type>line</type>
        <x>d-type|Output|x1</x>
        <y>d-type|Output|y1</y>
        <xlabel>label X</xlabel>
        <ylabel>label Y</ylabel>
        <gridLocation>
          <x>0 2</x>
          <y>0</y>
        </gridLocation>
      </plot>
      <plot>
        <type>scatter</type>
        <x>d-type|Output|x2</x>
        <y>d-type|Output|y2</y>
        <xlabel>label X</xlabel>
        <ylabel>label Y</ylabel>
        <gridLocation>
          <x>0 2</x>
          <y>1</y>
        </gridLocation>
      </plot>
```

```
      </plot>
    </plotSettings>
  </Plot>
</OutStreams>
```

The axis labels are conditionally optional nodes that can be defined under the <plotSetting>. If the plot does not contain any sub-plots, i.e. <gridSpace> is not defined then the axis labels are global parameters for the figure which are deifned under <plotSettings>, otherwise the axis labels can be defined under <plot> for each sub-plot seperately.

- <xlabel>, string, optional parameter, the x axis label.
- <ylabel>, string, optional parameter, the y axis label.
- <zlabel>, string, optional parameter (3D plots only), the z axis label.

As already mentioned, within the <plotSettings> block, at least a <plot> sub-block needs to be specified. Independent of the plot type, some keywords are mandatory:

- <type>, string, required parameter, the plot type (for example, line, scatter, wireframe, etc.).
- <x>, string, required parameter, this parameter needs to be considered as the x coordinate.
- <y>, string, required parameter, this parameter needs to be considered as the y coordinate.
- <z>, string required parameter (3D plots only), this parameter needs to be considered as the z coordinate.

In addition, other plot-dependent keywords, reported in Section 16.2.1.3, can be provided.

Under the <plot> sub-block other optional keywords can be specified, such as:

- <xlabel>, string, optional parameter, the x axis label.
- <ylabel>, string, optional parameter, the y axis label.
- <zlabel>, string, optional parameter (3D plots only), the z axis label.
- <gridLocation>, xmlNode, optional xmlNode (depending on the grid geometry)

- <x>, integer, required parameter, the position of the subPlot in the grid Space. if this node has a single value then the subplot occupies a single node at the specified location, otherwise the second integer represents the number of nodes that this subplot occupies, i.e. in the example above the first subplot occupies 2 nodes starting from the zero node in x direction.
- <y>, integer, required parameter, the position of the subPlot in the grid Space. if this node has a single value then the subplot occupies a single node at the specified location, otherwise the second integer represents the number of nodes that this subplot occupies, i.e. in the example above the first subplot occupies a single node at the zero node in y direction.

- <colorMap>, string, optional parameter, this parameter will be used to define a color map.

As already mentioned, the Plot system accepts as input for the visual parameters (i.e., x, y, z, colorMap), data only from a DataObjects object. Considering the structure of "DataObjects," the parameters are inputted as follows: `DataObjectName|Input(or)Output|variableName`.

If the "variableName" contains the symbol `|`, it must be surrounded by brackets: `[DataObjectName|Inpu`

16.2.1.3 Predefined Plotting System: 2D/3D

As already mentioned above, the Plotting system provides a specialized input structure for several different kind of plots specified in the <type> node:

- 2 Dimensional plots:
  - `scatter` creates a scatter plot of x vs y, where x and y are sequences of numbers of the same length.
  - `line` creates a line plot of x vs y, where x and y are sequences of numbers of the same length.
  - `histogram` computes and draws the histogram of x. Note: This plot accepts only the XML node <x> even if it is considered as a 2D plot type.
  - `stem` plots vertical lines at each x location from the baseline to y, and places a marker there.
  - `step` creates a 2 dimensional step plot.
  - `pseudocolor` creates a pseudocolor plot of a two dimensional array. The two dimensional array is built creating a mesh from <x> and <y> data, in conjunction with the data specified in the <colorMap> node.

- `contour` builds a contour plot creating a plot from <x> and <y> data, in conjunction with the data specified in the <colorMap> node.
- `filledContour` creates a filled contour plot from <x> and <y> data, in conjunction with the data specified in the <colorMap> node.

- 3 Dimensional plots:
  - `scatter` creates a scatter plot of (x,y) vs z, where x, y, z are sequences of numbers of the same length.
  - `line` creates a line plot of (x,y) vs z, where x, y, z are sequences of numbers of the same length.
  - `stem` creates a 3 Dimensional stem plot of (x,y) vs z.
  - `surface` creates a surface plot of (x,y) vs z. By default it will be colored in shades of a solid color, but it also supports color mapping.
  - `wireframe` creates a 3D wire-frame plot. No color mapping is supported.
  - `tri-surface` creates a 3D tri-surface plot. It is a surface plot with automatic triangulation.
  - `contour3D` builds a 3D contour plot creating the plot from <x>, <y> and <z> data, in conjunction with the data specified in <colorMap>.
  - `filledContour3D` builds a filled 3D contour plot creating the plot from <x>, <y> and <z> data, in conjunction with the data specified in <colorMap>.
  - `histogram` computes and draws the histogram of x and y. Note: This plot accepts only the XML nodes <x> and <y> even if it is considered as 3D plot type since the frequency is mapped to the third dimension.

As already mentioned, the settings for each plot type are specified within the XML block called <plot>. The sub-nodes that are available depends on the plot type as each plot type has its own set of parameters that can be specified.

In the following sub-sections all the options for the plot types listed above are reported.

16.2.2 2D & 3D Scatter plot

In order to create a "scatter" plot, either 2D or 3D, the user needs to write in the <type> body the keyword "scatter." In order to customize the plot, the user can define the following XML sub nodes:

- <s>, integer, optional field, represents the size in points$\hat{2}$. The "points" have the same meaning of the font size (e.g. Times New Roman, pts 10). In here the user specifies the area

of the marker size.
Default: 20

- <c>, string, optional field, specifies the color or sequence of color to use. <c> can be a single color format string, a sequence of color specifications of length N, or a sequence of N numbers to be mapped to colors using the cmap and norm specified via <kwargs>. Note: <c> should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. <c> can be a 2D array in which the rows are RGB or RGBA. Note: <colorMap> will overwrite <c>. If <colorMap> is defined then the color set used can be defined by <cmap>. If no <cmap> is given then the default color set of "matplotlib.pyplot.scatter" method in [4] is used. If <colorMap> is not defined then the plot is in solid color (default blue) as defined with <color> in <kwargs>.
- <marker>, string, optional field, specifies the type of marker to use.
  Default: o
- <alpha>, string, optional field, sets the alpha blending value, between 0 (transparent) and 1 (opaque).
  Default: None
- <linewidths>, string, optional field, widths of lines used in the plot. Note that this is a tuple, and if you set the linewidths argument you must set it as a sequence of floats.
  Default: None;
- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.scatter" method in [4].

16.2.3 2D & 3D Line plot

In order to create a "line" plot, either 2D or 3D, the user needs to write in the <type> body the keyword "line." In order to customize the plot, the user can define the following XML sub nodes:

- <interpolationType>, string, optional field, is the type of interpolation algorithm to use for the data. Available options are "nearest," "linear," "cubic," "multiquadric," "inverse,"

"gaussian," "Rbflinear," "Rbfcubic," "quintic," and "thin_plate."
Default: linear

- <interpPointsX>, integer, optional field, sets the number of points need to be used for interpolation of the x axis.
- <interpPointsY>, integer, optional field, sets the number of points need to be used for interpolation of the y axis. (only 3D line plot). Note: If <colorMap> is used then a scatter plot will be plotted.

16.2.4 2D & 3D Histogram plot

In order to create a "histogram" plot, either 2D or 3D, the user needs to write in the <type> body the keyword "histogram." In order to customize the plot, the user can define the following XML sub nodes:

- <bins>, integer or array_like, optional field, sets the number of bins if an integer is used or a sequence of edges if a python list is used.
  Default: 10
- <normed>, boolean, optional field, if True then the the histogram will be normalized to 1.
  Default: False
- <weights>, sequence, optional field, represents an array of weights, of the same shape as x. Each value in x only contributes its associated weight towards the bin count (instead of 1). If normed is True, the weights are normalized, so that the integral of the density over the range remains 1.
  Default: None
- <cumulative>, boolean, optional field, if True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values. The last bin gives the total number of data points. If normed is also True then the histogram is normalized such that the last bin equals 1. If cumulative evaluates to less than 0 (e.g., -1), the direction of accumulation is reversed. In this case, if normed is also True, then the histogram is normalized such that the first bin equals 1.
  Default: False
- <histtype>, string, optional field, The type of histogram to draw:
  - bar is a traditional bar-type histogram. If multiple data sets are given the bars are arranged side by side.
  - barstacked is a bar-type histogram where multiple data sets are stacked on top of each other.

  - step generates a line plot that is by default unfilled.
  - stepfilled generates a line plot that is by default filled.

  Default: bar

- <align>, string, optional field, controls how the histogram is plotted.
  - left bars are centered on the left bin edge.
  - mid bars are centered between the bin edges.
  - right bars are centered on the right bin edges.

  Default: mid

- <orientation>, string, optional field, specifies the orientation of the histogram:
  - horizontal
  - vertical

  Default: vertical

- <rwidth>, float, optional field, sets the relative width of the bars as a fraction of the bin width.
  Default: None

- <log>, boolean, optional field, sets a log scale.
  Default: False

- <color>, string, optional field, specifies the color of the histogram.
  Default: blue;

- <stacked>, boolean, optional field, if True, multiple data elements are stacked on top of each other. If False, multiple data sets are aranged side by side if histtype is 'bar' or on top of each other if histtype is 'step.'
  Default: False

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2>[56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.hist" method in [4].

16.2.5 2D & 3D Stem plot

In order to create a "stem" plot, either 2D or 3D, the user needs to write in the <type> body the keyword "stem." In order to customize the plot, the user can define the following XML sub nodes:

- <linefmt>, string, optional field, sets the line style used in the plot.
  Default: b-
- <markerfmt>, string, optional field, sets the type of marker format to use in the plot.
  Default: bo
- <basefmt>, string, optional field, sets the base format.
  Default: r-
- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.).

  For reference regarding the available kwargs, see "matplotlib.pyplot.stem" method in [4].

16.2.6 2D Step plot

In order to create a 2D "step" plot, the user needs to write in the <type> body the keyword "step." In order to customize the plot, the user can define the following XML sub nodes:

- <where>, string, optional field, specifies the positioning:
  - pre, the interval from x[i] to x[i+1] has level y[i+1]
  - post, that interval has level y[i]

- mid, the jumps in y occur half-way between the x-values

Default: mid

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.step" method in [4].

16.2.7 2D Pseudocolor plot

In order to create a 2D "pseudocolor" plot, the user needs to write in the <type> body the keyword "pseudocolor." In order to customize the plot, the user can define the following XML sub nodes:

- <interpolationType>, string, optional field, is the type of interpolation algorithm to use for the data. Available options are "nearest," "linear," "cubic," "multiquadric," "inverse," "gaussian," "Rbflinear," "Rbfcubic," "quintic," and "thin_plate."
Default: [linear]

- <interpPointsX>, integer, optional field, sets the number of points need to be used for interpolation of the x axis.

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.pcolor" method in [4].

16.2.8 2D Contour or filledContour plots

In order to create a 2D "contour" or "filledContour" plot, the user needs to write in the <type> body the keyword "contour" or "filledContour," respectively. In order to customize the plot, the user can define the following XML sub-nodes:

- <numberBins>, integer, optional field, sets the number of bins.
  Default: 5
- <interpolationType>, string, optional field, is the type of interpolation algorithm to use for the data. Available options are "nearest," "linear," "cubic," "multiquadric," "inverse," "gaussian," "Rbflinear," "Rbfcubic," "quintic," and "thin_plate."
  Default: linear
- <interpPointsX>, integer, optional field, sets the number of points need to be used for interpolation of the x axis.
- <colorMap> vector is the array to visualize. If <colorMap> is defined then the color set used can be defined by <cmap>. If no <cmap> is given then the plot is in solid color (default blue) as defined with <color> in <kwargs>.
- <cmap>, string, optional field, defines the color map to use for this plot.
  Default: None
- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.contour" method in [4].

16.2.9 3D Surface Plot

In order to create a 3D "surface" plot, the user needs to write in the <type> body the keyword "surface." In order to customize the plot, the user can define the following XML sub nodes:

- <rstride>, integer, optional field, specifies the array row stride (step size).
  Default: 1

- <cstride>, integer, optional field, specifies the array column stride (step size).
  Default: 1

- <cmap>, string, optional field, defines the color map to use for this plot.
  Default: None Note: If <colorMap> is defined then the plot will always use a color set even if no <cmap> is given. In such a case, if no <cmap> is given, then the default color set of "matplotlib.pyplot.surface" method in [4] is used. If <colorMap> and <cmap> are both not defined then the plot is in solid color (default blue) as defined with <color> in <kwargs>.

- <antialiased>, boolean, optional field, determines whether or not the rendering should be antialiased.
  Default: False

- <linewidth>, integer, optional field, defines the widths of lines rendered on the plot.
  Default: 0

- <interpolationType>, string, optional field, is the type of interpolation algorithm to use for the data. Available options are "nearest," "linear," "cubic," "multiquadric," "inverse," "gaussian," "Rbflinear," "Rbfcubic," "quintic," and "thin_plate."
  Default: linear

- <interpPointsX>, integer, optional field, sets the number of points need to be used for interpolation of the x axis.

- <interpPointsY>, integer, optional field, sets the number of points need to be used for interpolation of the y axis.

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.surface" method in [4].

16.2.10 3D Wireframe Plot

In order to create a 3D "wireframe" plot, the user needs to write in the <type> body the keyword "wireframe." In order to customize the plot, the user can define the following XML sub nodes:

- <rstride>, integer, optional field, sets the array row stride (step size).
  Default: 1

- <cstride>, integer, optional field, sets the array column stride (step size).
  Default: 1

- <cmap>, string, optional field, defines the color map to use for this plot.
  Default: None Note: <cmap> is not applicable in the current version of MatPlotLib for wireframe plots. However, if the colorMap option is set then a surface plot is plotted with a transparency of 0.4 on top of wireframe to give a visual colormap. Note: If <colorMap> is defined then the plot will always use a color set even if no <cmap> is given. In such a case, if no <cmap> is given, then the default color set of "matplotlib.pyplot.surface" method in [4] is used. If <colorMap> and <cmap> are both not defined then the plot is in solid color (default blue) as defined with <color> in <kwargs>.

- <interpolationType>, string, optional field, is the type of interpolation algorithm to use for the data. Available options are "nearest," "linear," "cubic," "multiquadric," "inverse," "gaussian," "Rbflinear," "Rbfcubic," "quintic," and "thin_plate."
  Default: linear

- <interpPointsX>, integer, optional field, sets the number of points need to be used for interpolation of the x axis.

- <interpPointsY>, integer, optional field, sets the number of points need to be used for interpolation of the y axis.

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.wireframe" method in [4].

16.2.11 3D Tri-surface Plot

In order to create a 3D "tri-surface" plot, the user needs to write in the <type> body the keyword "tri-surface." In order to customize the plot, the user can define the following XML sub nodes:

- <color>, string, optional field, sets the color of the surface patches.
  Default: b

- <shade>, boolean, optional field, determines whether to apply shading or not.
  Default: False

- <cmap>, string, optional field, defines the color map to use for this plot.
  Default: None Note: If <colorMap> is defined then the plot will always use a color set even if no <cmap> is given. In such a case, if no <cmap> is given, then the default color set of "matplotlib.pyplot.trisurface" method in [4] is used. If <colorMap> and <cmap> are both not defined then the plot is in solid color (default blue) as defined with <color> in <kwargs>.

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.trisurface" method in [4].

16.2.12 3D Contour or filledContour plots

In order to create a 3D "Contour" or "filledContour" plot, the user needs to write in the <type> body the keyword "contour3D" or "filledContour3D," respectively. In order to customize these plots, the user can define the following XML sub nodes:

- <numberBins>, integer, optional field, sets the number of bins to use.
  Default: 5

- <interpolationType>, string, optional field, is the type of interpolation algorithm to use for the data. Available options are "nearest," "linear," "cubic," "multiquadric," "inverse," "gaussian," "Rbflinear," "Rbfcubic," "quintic," and "thin_plate."
  Default: linear

- <interpPointsX>, integer, optional field, sets the number of points need to be used for interpolation of the x axis.

- <interpPointsY>, integer, optional field, sets the number of points need to be used for interpolation of the y axis.
- <colorMap> vector is the array to visualize. If <colorMap> is defined then the color set used can be defined by <cmap>. If no <cmap> is given then the plot is in solid color (default blue) as defined with <color> in <kwargs>.
- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.). For reference regarding the available kwargs, see "matplotlib.pyplot.contour3d" method in [4].

16.2.13 DataMining plots

In order to create a "DataMining" plot, the user needs to write in the <type> body the keyword "dataMining". "DataMining" plots are based on 2D or 3D Scattering plots, depending on the method/algorithm used in the "DataMining" postprocessor [see 17.5.9]. These plots are created to ease the color labeling the clusters, etc parameters in the data. The following are the optional or required input parameters that can be used in these plots additional to the coordinate inputs <x>, <y>, or <z> depending on the dimension:

- <type>, string, required field, this block should read "dataMining" in order to create a data mining plot.
- <SKLtype>, string, required field, name of the algorithm used in the "dataMining" post-processor. It is one of:
  - cluster: for clustering algorithms, such as KMeans clustering.
  - bicluster (Note: not implemented yet!)
  - mixture: for Gaussian mixture algorithms, such as GMM classifier
  - manifold: for Manifold Learning algorithms, such as Spectral Embedding
  - decomposition: for decomposing signals in components algorithms, such as Principal Component Analysis (PCA)

- <clusterLabels>, string, optional field, defines the place where the labels of the clusters are located. As in the visual parameters (i.e., x,y,z and colorMap) this is also from a DataObjects object. Considering the structure of "DataObjects", the labels inputted as follows: `DataObjectName|Output|DataMiningPPNameLabels`.
  Default: None

- <noClusters>, integer, optional field, defines the number of clusters used in the "dataMining" postprocessor
  Default: 1

- <kwargs>, within this block the user can specify optional parameters with the following format:

```
<kwargs>
 <param1>value1</param1>
 <param2>value2</param2>
</kwargs>
```

  The kwargs block is able to convert whatever string into a python type (for example <param1> {'1stKey':45}</param1> will be converted into a dictionary, <param2> [56,67] </param2> into a list, etc.).

  For reference regarding the other available kwargs, see "matplotlib.pyplot.scatter" method in [4].

16.2.14 Example XML input

```
<OutStreams>
  <Plot name='2DHistoryPlot' interactive='False'
     overwrite='False'>
    <actions>
      <how>pdf,png,eps</how>
      <title>
        <text>***</text>
      </title>
    </actions>
    <plotSettings>
       <plot>
        <type>line</type>
        <x>stories|Output|time</x>
        <y>stories|Output|pipe1_Hw</y>
        <kwargs>
         <color>green</color>
         <label>pipe1-Hw</label>
```

```
        </kwargs>
      </plot>
       <plot>
        <type>line</type>
        <x>stories|Output|time</x>
        <y>stories|Output|pipe1_aw</y>
        <kwargs>
         <color>blue</color>
         <label>pipe1-aw</label>
        </kwargs>
      </plot>
      <xlabel>time [s]</xlabel>
      <ylabel>evolution</ylabel>
    </plotSettings>
  </Plot>
</OutStreams>
```