Informationen

Entwicklungsumgebung

Spyder ist gut, wenn man Plots direkt in der IDE darstellen möchte, jedoch hängte sich die IDE am Tag 2-4 Mal auf. Des Weiteren war die Vervollständigung nicht gut. Es gab auch Fälle, in denen sich Spyder beim Starten eines Programms sich aufgehängt hat und ein Neustart vonnöten war.

PyCharm hingegen hat eine gute automatische Vervollständigung und das Starten eines Python Programms ist einfach. Eine direkte Unterstützung für das Anzeigen von Graphen in der DIE habe ich zwar nicht gefunden, jedoch ist das, wenn man Widgets für Orange erstellen gar nicht notwendig, weil, wenn man einen Graphen darzustellen hat, dies höchstwahrscheinlich sowieso im erstellten Orange Widget getätigt wird. Als verwendete Python Version kann die von Anaconda hergenommen werden. Generell ist auch die Erkennung und das Anzeigen des Typen einer Variable wesentlich besser umgesetzt.

PyQtGraph

Skalierung

Es gibt anscheinend einen Bug, was die Skalierungseinstellung unter Windows betrifft, wenn man 2 Monitore mit verschiedenen Skalierungseinstellungen verwendet.

So kommt es dazu, dass die Skalierung eines Graphs inkorrekt dargestellt wird. Dies betrifft hauptsächlich die Y-Achse, jedoch auch zum Teil die X-Achse.

Das Problem trat bei mir auf, als ich einen Laptop und externen Monitor verwendet habe, die eine unterschiedliche Skalierungseinstellung hatten. Das Umstellen der Skalierung hat das Problem gelöst. Dazu anzumerken ist, dass nicht auf beide Bildschirme ein Graph falsch dargestellt wurden, sondern nur auf einem der beiden.

Mehrere Plots

PyQtGraph bietet ein eigenes Widget GraphicsLayoutWidget, wenn man mehrere Plots darstellen möchte. Mittels addPlot kann ein weiterer Plot hinzugefügt werden.

Wenn man einem Plot anschließend Daten übergeben möchte, macht man dies mittels der Funktion plot. Möchte man etwas Neues im Plot Zeichnen, so ist zuvor clear aufzurufen, bevor man wieder plot aufruft. Soweit ich weiß, gäbe es auch eine Funktion setData, die höchstwahrscheinlich den Aufruf von clear erspart und dasselbe wie plot() macht.

Mehrere Y-Achsen:

In PyQtGraph ist es möglich, dass ein Plot mehrere Y-Achsen besitzt, anscheinend wird dies mit mehreren Viewboxen umgesetzt. PyQtGraph stellt bezüglich mehrerer Y-Achsen ein Beispiel bereit.

Region

PyQtGraph ermöglicht es in einem Graphen eine Region zu erstellen. Den ausgewählten Bereich kann man innerhalb eines weiteren Graphen darstellen lassen. Somit ist es möglich, dass man nur einen gewissen Bereich eines Graphens näher betrachten kann.

DataAxisItem

Damit man Zeiten darstellen kann, muss man DataAxisItem verwenden. Jedoch gibt es die benötigte Klasse anscheinend noch nicht in der aktuell verwendeten Version. Deswegen muss der Sourcecode für die Klasse fürs Erste herauskopiert werden, dies ist später höchstwahrscheinlich nicht mehr notwendig.

Beim Darstellen der Zeit ist die Anzahl der Stunden nicht korrekt. Woran das liegt, habe ich nicht herausgefunden.

In PyQt kann man bei einem Graphen auch einen Datacursor erstellen, dies kann man machen, indem man auf einen Mouseevent reagiert und eine Callback Funktion dafür hinterlegt.

Dadurch kann man den Wert der Position in zum Beispiel einem Label anzeigen lassen. Bei den Beispielen zu PyQtGraph gibt es eines, indem das verwendet wird.

Orange

GridLayout

Es ist möglich, dass man bei den 2 vorgegeben Bereichen, die Orange bei einem Widget vorgibt, ein Gridlayout verwendet.

Folgender Code zeigt wie dies umgesetzt werden kann:

self.controlArea = QWidget(self.controlArea)

self.layout().addWidget(self.controlArea)

layout = QGridLayout()

self.controlArea.setLayout(layout)



Warning, Error, Info

In Orange kann eine Warnung, Error als auch Info angezeigt werden.

Bei den Beispielen, die ich bisher gesehen habe, wurde innerhalb einer Klasse weitere Klassen (Error, Warning, Info) erstellt. Diese Klassen hatten variablen des Typs Msg. Des Weiteren muss die innere Klasse von OWWidget.Warning, OWWidget.Error oder OWWidget.Info erben.

Die Variablen waren namhaft benannt und der Msg wurde im Konstruktor die entsprechende Benachrichtigung übergeben.

Falls eine Notwendigkeit einer Benachrichtigung auftreten soll, so kann man die Sichtbarkeit der entsprechenden Benachrichtigung anstellen. Dies macht man, in dem man die Variable aufruft.

Beispiel:

class Error(OWWidget.Error):

unsupported\_date\_format = Msg(“the given date format is not supported”)

# So kann man die Benachrichtigung anzeigen lassen

Self.error.unsupported\_date\_format(shown = True)

# Möchte man die Benachrichtigung nicht mehr angezeigt haben, so kann man shown auf False #setzen.

Table

In Orange befinden sich die einzelnen Zeilen, die ein Table hat, innerhalb der Variablen attributes und metas. Innerhalb von metas sind die Spalten, die als Meta gekennzeichnet sind.

Wenn man überprüfen möchte, ob sich eine gewisse Spalte innerhalb einer Table befindet, so ist metas als auch attributes zu überprüfen.

Ein Table in Orange ist nicht veränderlich, möchte man eine neue Table mit weiteren Spalten erstellen, so ist eine neue Domain anzulegen mit den weiteren Spalten.

Mittels einer Funktion kann eine neue Table anschließend erstellt werden und auch die Werte eines alten Table übergeben werden.

Einbinden einer anderen Python Datei in einem Widget

Dies hat seltsamerweise beim Testen in der IDE gut funktioniert, jedoch wird das Widget, wenn es eine Python Datei, die im selben Verzeichnis ist, importiert, nicht in Orange angezeigt. Das Ausführen des Widgets in der IDE funktioniert jedoch.

Timeseries

Bei den Widgets, die eine Timeseries übergeben haben möchten, ist es nicht notwendig, bei einem Widget 2 Outputs (Table, Timeseries) bereitzustellen, weil, bei den Widgets, die eine Timeseries benötigen, auch eine Table entgegennehmen und diese intern in eine Timeseries umwandeln.

Seaborn

An sich kann Seaborn in Orange verwendet werden, weil es Widgets in Qt gibt, die dies ermöglichen, jedoch, erstellt meines Wissens nach Seaborn eine neue Figure und dies macht das Einbinden umständlich.

PyQt

Wenn man auf einen Event reagieren möchte, so kann man dies machen, indem man die Funktion connect verwendet. Mit dieser kann man ein Event, das bei dem entsprechenden Objekt auftreten kann, eine Funktion „hinterlegen“, die aufgerufen wird, wenn das Event eintrifft.

Es ist auch möglich, dass man ein 2. Fenster öffnet, dies kann man machen, indem man eine Klasse erstellt, die von QtGui.QMainWindow erbt.

Dem Konstruktor muss das OrangeWidget (self) übergeben werden.

Siehe hierfür das Beispiel qt\_second\_window.py

Häufig findet man im Internet, dass Klassen von QtGui verwendet, jedoch gibt es diese Klassen anscheinend nicht in der aktuellen Version von Qt unter QtGui zu finden.

Stattdessen ist QtWidgets zu verwenden.

Zum Beispiel: QtWidgets.QWidget

Matplotlib

Matplotlib scheint nicht schlecht zu sein, wenn man innerhalb einer IDE schnell einen Plot betrachten möchte, jedoch ist es anscheinend eher ungeeignet, wenn man es innerhalb eines Widgets verwenden möchte.

Einige triviale Aufgaben erwiesen sich mit Matplotlib als sehr umständlich. So muss man zum Beispiel die Funktion subplots() verwenden, wenn man mehrere Plots haben möchte. Dieser Funktion übergibt man, wie viele Reihen und Zeilen man haben möchte. Je nach Anzahl der Reihen und Zeilen unterscheidet sich die Rückgabetyp der Funktion subplots(). So muss man als Programmierer darauf achten, was man zurückgegeben bekommt und dies in einen einheitlichen Typ umwandeln, um die Rückgabe überhaupt gut weiter verwenden zu können.

Des Weiteren war es mir unklar, wie man die erstellten Plots mittels subplots wieder entfernen kann, bzw. wie man nur einen Plot entfernen bzw. hinzufügen kann.

Hier erweist sich PyQtGraph als wesentlich besser, vor allem, weil es hierfür sogar ein eigenes Widget anbietet.

Die Versuche/Beispiele findet man auf:

Die Beispiele von PyQt Graph findet man auf:

Sonstige Links:

Scrollbar:

<https://doc.qt.io/archives/qt-4.8/qscrollarea.html>

Kommunikation zwischen Fenster:

<https://www.youtube.com/watch?v=wOxzhX0QnAw>

Orange Errors:

<https://orange3.readthedocs.io/projects/orange-development/tutorial-utilities.html>

Orange Testing:

<https://orange3.readthedocs.io/projects/orange-development/testing.html>

Matplotlib Plots:

<https://matplotlib.org/gallery/user_interfaces/embedding_in_qt_sgskip.html#sphx-glr-gallery-user-interfaces-embedding-in-qt-sgskip-py>

Matplotlib Axes:

<https://matplotlib.org/3.1.1/gallery/ticks_and_spines/multiple_yaxis_with_spines.html>

<https://stackoverflow.com/questions/9103166/multiple-axis-in-matplotlib-with-different-scales>

Matplotlib Legend:

<https://matplotlib.org/3.1.1/gallery/event_handling/legend_picking.html>

Seaborn:

<https://stackoverflow.com/questions/41671867/embedding-figure-type-seaborn-plot-in-pyqt-pyqtgraph>

<https://github.com/diego0020/correlation_viewer>

<https://seaborn.pydata.org/examples/scatterplot_matrix.html>

<https://stackoverflow.com/questions/4652439/is-there-a-matplotlib-equivalent-of-matlabs-datacursormode>

Matplotlib Eventhandling:

<https://matplotlib.org/3.1.1/users/event_handling.html>

Matplotlib Subplots:

<https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html?highlight=multiple%20plot>

PyQt drop down button:

<https://pythonprogramming.net/drop-down-button-window-styles-pyqt-tutorial/>

PyQt SQL:

<https://www.tutorialspoint.com/pyqt/pyqt_database_handling.htm>

PyQt Table:

<https://doc.qt.io/qt-5/qtablewidget.html>

PyQt Scatterplot:

<https://github.com/pyqtgraph/pyqtgraph/blob/master/pyqtgraph/widgets/ScatterPlotWidget.py>

Python Performance:

<https://wiki.python.org/moin/PythonSpeed/PerformanceTips>