mpl$_{qt_viz}$Documentation

Release 1.0.11.dev5+g2fdc3ca

Nick Anthony

Oct 21, 2021

CONTENTS

CHAPTER

ONE

MPL_QT_VIZ

A Python package providing enhanced data visualization and ROI selection built on top of Matplotlib and PyQt5.

1.1 Subpackages

roiSelection	Useful classes for interacting with Matplotlib plots.
visualizers	Qt Widgets for interactive data visualization.

1.1.1 mpl_qt_viz.roiSelection

Useful classes for interacting with Matplotlib plots. Mostly for the purpose of drawing ROIs.

ROI Creators

EllipseCreator(ax, image[, onselect])	Allows the user to select an elliptical region.
LassoCreator(ax, image[, onselect])	Allows the user to select a region with freehand drawing.
RegionalPaintCreator(ax, im[, onselect])	A widget allowing the user to select a rectangular region with a bright region in it such as a fluorescent nucleus.
SquareCreator(ax, image[, onselect, sideLength])	
FullImPaintCreator(ax, im[, onselect])	Uses adaptive thresholding in an attempt to highlight all bright selectable regions in a fluorescence image.
WaterShedPaintCreator(ax, im[, onselect])	Uses Watershed technique in an attempt to highlight all bright selectable regions in a fluorescence image.

mpl_qt_viz.roiSelection.EllipseCreator

class mpl_qt_viz.roiSelection.**EllipseCreator**(*ax*, *image*, *onselect=None*)

Bases: mpl_qt_viz.roiSelection._creatorWidgets._base.CreatorWidgetBase

Allows the user to select an elliptical region.

Parameters

- **ax** (*Axes*) – The matplotlib *Axes* that you want to interact with.
- **image** (*AxesImage*) – A reference to a matplotlib *AxesImage*. Selectors may use this ref-

erence to get information such as data values from the image for computer vision related tasks.

- **onselect** (*typing.Callable*) – A callback that will be called when the user hits 'enter'. Should have signature (polygonCoords, sparseHandleCoords).

static getHelpText()
Return a description of the selector which can be used as a tooltip.

reset()
Reset the state of the selector so it's ready for a new selection.

mpl_qt_viz.roiSelection.LassoCreator

class mpl_qt_viz.roiSelection.**LassoCreator**(*ax*, *image*, *onselect=None*)
Bases: mpl_qt_viz.roiSelection._creatorWidgets._base.CreatorWidgetBase

Allows the user to select a region with freehand drawing.

Parameters

- **ax** (*Axes*) – A reference to the matplotlib *Axes* that this selector widget is active on.
- **image** (*AxesImage*) – A reference to a matplotlib *AxesImage*. Selectors may use this reference to get information such as data values from the image for computer vision related tasks.
- **onselect** – A callback function that will be called when the selector finishes a selection.

static getHelpText()
Return a description of the selector which can be used as a tooltip.

reset()
Reset the state of the selector so it's ready for a new selection.

mpl_qt_viz.roiSelection.RegionalPaintCreator

class mpl_qt_viz.roiSelection.**RegionalPaintCreator**(*ax*, *im*, *onselect=None*)
Bases: mpl_qt_viz.roiSelection._creatorWidgets._base.CreatorWidgetBase

A widget allowing the user to select a rectangular region with a bright region in it such as a fluorescent nucleus. Otsu thresholding will then be used to draw an ROI on the bright region.

Args: ax: A reference to the matplotlib *Axes* that this selector widget is active on. image: A reference to a matplotlib *AxesImage*. Selectors may use this reference to get information such as data values from the image

for computer vision related tasks.

onselect: A callback function that will be called when the selector finishes a selection.

findContours(*rect*)
Detect bright regions within the specified rectangle and draw them.

Parameters **rect** (Rectangle) – A matplotlib *Rectangle* used to specify the search region of the image for bright regions.

static getHelpText()
Return a description of the selector which can be used as a tooltip.

reset()
Reset the state of the selector so it's ready for a new selection.

mpl_qt_viz.roiSelection.SquareCreator

class mpl_qt_viz.roiSelection.**SquareCreator**(*ax*, *image*, *onselect=None*, *sideLength=5*)
Bases: mpl_qt_viz.roiSelection._creatorWidgets._base.CreatorWidgetBase

static getHelpText()
Return a description of the selector which can be used as a tooltip.

reset()
Reset the state of the selector so it's ready for a new selection.

mpl_qt_viz.roiSelection.FullImPaintCreator

class mpl_qt_viz.roiSelection.**FullImPaintCreator**(*ax*, *im*, *onselect=None*)
Bases: mpl_qt_viz.roiSelection._creatorWidgets._base.CreatorWidgetBase

Uses adaptive thresholding in an attempt to highlight all bright selectable regions in a fluorescence image.

Parameters

- **ax** (*Axes*) – The matplotlib *Axes* that you want to interact with.
- **im** (*AxesImage*) – A reference to a matplotlib *AxesImage*. The data from this object is used to detect bright regions.
- **onselect** – A callback that will be called when the user hits 'enter'. Should have signature (polygonCoords, sparseHandleCoords).

static getHelpText()
Return a description of the selector which can be used as a tooltip.

paint(*forceRedraw=True*)
Refresh the detected regions.

Parameters **forceRedraw** (bool) – If *True* then polygons will be cleared and redrawn even if we don't detect that our status is *stale*

reset()
Reset the state of the selector so it's ready for a new selection.

set_active(*active*)
Set whether the widget is active.

mpl_qt_viz.roiSelection.WaterShedPaintCreator

class mpl_qt_viz.roiSelection.**WaterShedPaintCreator**(*ax*, *im*, *onselect=None*)
Bases: mpl_qt_viz.roiSelection._creatorWidgets._base.CreatorWidgetBase

Uses Watershed technique in an attempt to highlight all bright selectable regions in a fluorescence image.

Parameters

- **ax** (*Axes*) – The matplotlib *Axes* that you want to interact with.
- **im** (*AxesImage*) – A reference to a matplotlib *AxesImage*. The data from this object is used to detect bright regions.

- **onselect** – A callback that will be called when the user hits 'enter'. Should have signature (polygonCoords, sparseHandleCoords).

static getHelpText()
Return a description of the selector which can be used as a tooltip.

paint(*forceRedraw=True*)
Refresh the detected regions.

Parameters **forceRedraw** (bool) – If *True* then polygons will be cleared and redrawn even if we don't detect that our status is *stale*

reset()
Reset the state of the selector so it's ready for a new selection.

set_active(*active*)
Set whether the widget is active.

Utility

AdjustableSelector(ax, image, selectorClass)	This class manages an roi selector.
PolygonModifier(ax[, onselect, onCancelled])	A polygon editor. https://matplotlib.org/gallery/event_handling/poly_editor.html Key-bindings: 'd' delete the vertex under point 'i' insert a vertex at point. You must be within epsilon of the line connecting two existing vertices.

mpl_qt_viz.roiSelection.AdjustableSelector

class mpl_qt_viz.roiSelection.**AdjustableSelector**(*ax*, *image*, *selectorClass*, *onfinished=None*, *onPolyTuningCancelled=None*)

Bases: object

This class manages an roi selector. By setting *adjustable* true then when the selector calls its *onselect* function the results will be passed on to a PolygonInteractor for further tweaking. Tweaking can be confirmed by pressing enter. at the end the selector will pass a set of coordinates to the *onfinished* function if it has been set.

Parameters

- **ax** (*Axes*) – A matplotlib *Axes* to interact with.
- **image** (*AxesImage*) – A matplotlib *AxesImage*. Some selectors use the data in this image for their selection.
- **selectorClass** (*typing.Type[CreatorWidgetBase]*) – A class that implements *SelectorWidgetBase*. This will be the intial selector used.
- **onfinished** (*typing.Optional[typing.Callable]*) – a callback function when the selection finished. The function should accept a single input argument which is a list of the 2d coordinated outlining the selected polygon.

property adjustable: bool
Determines whether or not the polygon interactor will be used to adjust the selection at the end of the initial selection.

Return type bool

finish(*verts*, *handles*)
This callback is registered with the selectorWidget when we are not in adjustable mode. In adjustable mode it is instead registered with the polygon adjuster. It deactivates the class and calls the *onfinished* callback.

reset()
Clear all artists used by the selector. :todo: Shouldn't this check if the *adjuster* is active and reset it as well?

setActive(*active*)
This activates the selector. for a looping selection you should call this method from the onfinished function.

setSelector(*selectorClass*)
Remove the current selector and replace it with a new type of selector.

Parameters **selectorClass** (Type) – A class the implements *SelectorWidgetBase*.

mpl_qt_viz.roiSelection.PolygonModifier

class mpl_qt_viz.roiSelection.**PolygonModifier**(*ax*, *onselect=None*, *onCancelled=None*)
Bases: mpl_qt_viz.roiSelection._modifierWidgets._base.ModifierWidgetBase

A polygon editor. https://matplotlib.org/gallery/event_handling/poly_editor.html Key-bindings:

'd' delete the vertex under point 'i' insert a vertex at point. You must be within epsilon of the

line connecting two existing vertices

Parameters

- **ax** (*Axes*) – A matplotlib *Axes* that you want to interact with.
- **onselect** (*typing.Optional[ModifierWidgetBase.SelectionFunction]*) – A callback that will be called when the user hits 'enter'. Should have signature (polygonCoords, sparseHandleCoords).

epsilon
The pixel distance required to detect a mouse-over event.

Type int

static getHelpText()
Return a description of the selector which can be used as a tooltip.

initialize(*handles*)
Given a set of points this will initialize the artists to them to begin modification.

Parameters **handles** (Sequence[Sequence[Tuple[float, float]]]) – A sequence of 2d coordinates to intialize the polygon to. Each point will become a draggable handle

1.1.2 mpl_qt_viz.visualizers

Qt Widgets for interactive data visualization.

Classes

MultiPlot(artists, title[, parent])	A widget that allows the user to flip through a set of matplotlib artists (images, plots, etc.)
PlotNd(data[, names, initialCoords, title, ...])	A convenient widget for visualizing data that is 3D or greater.
PlotNdCanvas(data, names[, initialCoords, ...])	The matplotlib canvas for the PlotND widget.
DockablePlotWindow([title])	

mpl_qt_viz.visualizers.MultiPlot

class mpl_qt_viz.visualizers.**MultiPlot**(*artists*, *title*, *parent=None*)

Bases: PyQt5.QtWidgets.QWidget

A widget that allows the user to flip through a set of matplotlib artists (images, plots, etc.)

Parameters

- **artists** (List[List[Artist]]) – A list of lists of matplotlib 'Artists`. each list will comprise a single frame, just like the matplotlib *ArtistAnimation* works.
- **title** (*str*) – The name for the title of the window

imshow(**args*, ***kwargs*)

Mirrors the pyplot.imshow function. Adds a new image to the set of images shown by this widget.

showNextIm()

Display the next set of display elements.

showPreviousIm()

Display the previous set of display elements.

mpl_qt_viz.visualizers.PlotNd

class mpl_qt_viz.visualizers.**PlotNd**(*data*, *names=None*, *initialCoords=None*, *title=''*, *parent=None*, *indices=None*, *flags=1*)

Bases: PyQt5.QtWidgets.QWidget

A convenient widget for visualizing data that is 3D or greater. This is a standalone widget which extends the functionality of *PlotNdCanvas*.

Parameters

- **data** (ndarray) – A 3D or greater numpy array of numeric values.
- **names** (Optional[Tuple[str, ...]]) – A sequence of labels for each axis of the data array.
- **initialCoords** (Optional[Tuple[int, ...]]) – An optional sequence of the coordinates to initially se the ND crosshair to. There should be one coordinate for each axis of the data array.
- **title** (Optional[str]) – A title for the window.
- **parent** (Optional[QWidget]) – The Qt Widget that serves as the parent for this widget.

- **indices** (Optional[Sequence[ndarray]]) – An optional tuple of 1d arrays of values to set as the indexes for each dimension of the data. Elements of the list can be set to *None* to skip setting a custom index for that dimension.
- **flags** – See the *flags* constructor argument for a QWidget. Default value is *Window*

data
A reference to the 3D or greater numpy array. This can be safely modified.

setColorMap(*cmap*)
Set the colormap used to display data.

Parameters **cmap** (Union[str, Colormap]) – This value will be have the same effect as the argument of Matplotlib's *AxesImage.set_cmap()*

setLimits(*Min*, *Max*)
Set the limits of the colormap.

Parameters

- **Min** (float) – The data value that will correspond to the minimum of the colormap.
- **Max** (float) – The data value that will correspond to the maximum of the colormap.

mpl_qt_viz.visualizers.PlotNdCanvas

class mpl_qt_viz.visualizers.**PlotNdCanvas**(*data*, *names*, *initialCoords=None*, *indices=None*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*)
Bases: matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg

The matplotlib canvas for the PlotND widget.

Parameters

- **data** (ndarray) – 3D or greater numeric data
- **names** (Tuple[str, ...]) – The names to label each dimension of the data with.
- **initialCoords** (Optional[Tuple[int, ...]]) – An optional tuple of coordinates to set the Nd crosshair to.
- **indices** (Optional[List]) – An optional tuple of 1d arrays of values to set as the indexes for each dimension of the data.

performBlit()
Re-render the axes efficiently using matplotlib *blitting*.

rollAxes()
Change the order of the axes of the data. Allows viewing the sideview of the data.

setAxesNames(*names*)
Set the names of to label each plot. :type names: Iterable[str] :param names: the order of the names should match the order of each corresponding axis in the data array.

setColorMap(*cmap*)
Set the colormap used to display data.

Parameters **cmap** (Union[str, Colormap]) – This value will be have the same effect as the argument of Matplotlib's *AxesImage.set_cmap()*

setIndices(*indices*)
Set the index values for each dimension of the array.

Parameters **indices** (Sequence[Sequence[float]]) – A list or tuple of index values for each dimension of the data array.

setSpectraViewActive(*active*)
Determines whether or not the Nd crosshair respons to mouse input. Allows us to disable the crosshair if we want the mouse to trigger other sorts of actions (e.g. ROI drawing)

updateLimits(*Max*, *Min*)
Update the range of values displayed. Similar to the set_clim method of a matplotlib image.

Parameters

- **Max** (float) – The maximum value displayed
- **Min** (float) – The minimum value displayed

updatePlots(*blit=True*)
This should be called after *self.coords* have been changed to update the data of each plot.

Parameters **blit** – If *True* then drawing will be done more efficiently through *blitting*. Sometimes this needs to be false to trigger a full redraw though.

mpl_qt_viz.visualizers.DockablePlotWindow

class mpl_qt_viz.visualizers.**DockablePlotWindow**(*title='Dockable Plots'*)
Bases: PyQt5.QtWidgets.QMainWindow

addFigure(*fig*, *title*, *dockArea='top'*)
Add a pre-existing Matplotlib Figure to a new dockable widget in the window.

Parameters

- **fig** (Figure) – A pre-existing Matplotlib Figure
- **title** (str) – The title for the new dockable widget
- **dockArea** (str) – The side of the window that the new plot should be initially placed in. If a figure has already been created on that side of the window then the new figure will be docked with the existing one. Accepted values are: 'left', 'right', 'top', and 'bottom'.

addWidget(*widget*, *title*, *dockArea='top'*)
Add any Qt widget to a dockable widget in our window.

Parameters

- **widget** (QWidget) – Any Qt Widget
- **title** (str) – The title of the new dockable widget.
- **dockArea** (str) – The dock area of the window to add the dockable widget to.

property figures: Dict[str, matplotlib.figure.Figure]
A dictionary of all the dockable figures in this window keyed by their titles.

Return type Dict[str, Figure]

subplots(*title*, *dockArea='top'*, *subplots_kwargs=None*, *subplot_kw=None*)
Create a new docked figure within the main window.

Parameters

- **title** (str) – The title for the new figure.

- **dockArea** (str) – The side of the window that the new plot should be initially placed in. If a figure has already been created on that side of the window then the new figure will be docked with the existing one. Accepted values are: 'left', 'right', 'top', and 'bottom'.
- **subplots_kwargs** (Optional[dict]) – This dictionary will be passed as the kwargs for *pyplot.subplots*
- **subplot_kw** (Optional[dict]) – This dictionary will be passed to the *subplot_kw* arg of *pyplot.subplots*

Returns The return values are the same as the return values of *pyplot.subplots*. Usually taking the form of (figure, axes).

CHAPTER

TWO

EXAMPLES

2.1 Examples

2.1.1 PlotNd visualization of a 3D array

```
from mpl_qt_viz.visualizers import PlotNd
import numpy as np
from PyQt5.QtWidgets import QApplication
import sys

# Generate a 3-dimensional dimension array with numpy.meshgrid.
# The Plot Nd Widget supports higher dimensionality as well.
x = np.linspace(0, 1, num=75)
y = np.linspace(0, 1, num=100)
z = np.linspace(0, 3, num=40)
X, Y, Z = np.meshgrid(x, y, z)
# Create a 3-dimensional example data array.
arr = np.sin(2 * np.pi * 1 * Z) + .5 * X + np.cos(2 * np.pi * 4 * Y)

#Run an application with the PlotNd widget
app = QApplication(sys.argv)
p = PlotNd(data=arr,
           names=('Dim1', 'D2', 'D3'),  # Manually sets how each dimension is labeled.
           indices=[y, x, z])  # Specifies the data range for each dimension.
p.setColorMap('plasma')
sys.exit(app.exec_())
```

2.1.2 DockablePlotWindow providing organization for related plots

```
import numpy as np
import random
from PyQt5.QtWidgets import QApplication
from mpl_qt_viz.visualizers import DockablePlotWindow
import sys

# Plot names that will be randomly selected from in this example
names = ['plot', 'data', 'other']
```

(continues on next page)

(continued from previous page)

```
# Valid plot location specifiers that will be randomly selected from in this example
areas = ['left', 'right', 'bottom', 'top']


def makePlot(ax):
    """Generate a random line plot on ax."""
    x = np.linspace(0, 10)
    # y = np.random.random(x.size)
    freq1 = .5 + np.random.rand()
    freq2 = .5 + np.random.rand()
    ax.plot(x, np.sin(freq1 * x))
    ax.plot(x, np.cos(freq2 * x), ls='--')


def makeImage(ax):
    """Generate a random image plot on ax."""
    freq = 1 + 10 * np.random.rand()
    X, Y = np.meshgrid(np.linspace(-1, 1), np.linspace(-1, 1))
    R = X**2 + Y**2
    arr = np.sin(freq * R)
    ax.imshow(arr, cmap='jet')

plotTypes = [('plot', makePlot), ('image', makeImage)]


app = QApplication([])  # Make an application for the widgets to run in.

w = DockablePlotWindow("My Dockable Plot Window")
for i in range(10):
    name, func = random.choice(plotTypes)
    # Use the widget's `subplots` method to generate matplotlib plots docked in the␣
↪widget.
    fig, ax = w.subplots(name, dockArea=random.choice(areas))
    func(ax)

w2 = DockablePlotWindow(title="My 2nd Plot Window")
for i in range(10):
    x = np.linspace(0, 1)
    y = np.random.random(x.size)
    fig, ax = w2.subplots(random.choice(names), dockArea=random.choice(areas))
    ax.plot(x, y)

sys.exit(app.exec())  # Run the application until all windows are closed
```

2.1.3 MultiPlot containing various images and line plots

```
import sys
import matplotlib.pyplot as plt
from PyQt5.QtWidgets import QApplication
import numpy as np
from mpl_qt_viz.visualizers import MultiPlot

app = QApplication(sys.argv)

# Generate a list of lists of artists and create a new MultiPlot with them.
ims = [[plt.imshow(np.random.random((512, 512))), plt.text(100, 100, str(i))] for i in␣
↪range(3)]
mp = MultiPlot(ims, "Images")

#Adjust the figure and axes
plt.gcf().subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0, hspace=0)
mp.ax.get_xaxis().set_visible(False)
mp.ax.get_yaxis().set_visible(False)
mp.show()  # Show the widget

#Create a second MultiPlot with line plots
fig, ax = plt.subplots()
lines = [ax.plot(np.random.random((50,))) for i in range(3)]
mp2 = MultiPlot(lines, 'Lines')
mp2.show()

sys.exit(app.exec())
```

todo Add example of ROI drawing tools.

Using the *PlotNd* widget to visualize hyperspectral imagery of a cancer cell *PlotNd* widget to visualize hyperspectral imagery of a cancer cell.

Using the `DockablePlotsWindow` to help organize a large number of plots.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

S

U

W