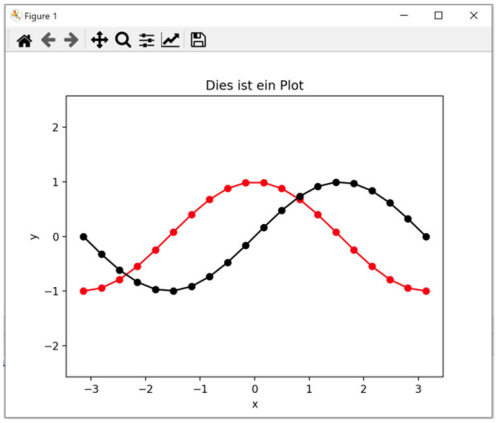
9 Matplotlib und PyQt5

9.1 Plotten mit Python Script

In Kapitel 2 haben wir bereits Matplotlib kennengelernt. Wir haben dazu Jupyter Lab verwendet um unsere Grafik darzustellen. Wir können das nun auch in einem Python-File versuchen:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2*np.pi, 20)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x,y1, "ro-")
plt.plot(x,y2, "ko-")

plt.title("Titel")
plt.xlabel("x")
plt.ylabel("y")

plt.axis("equal")
plt.show()
```

Wie wir sehen wird ein Fenster geöffnet und die Kurven werden geplottet. Dazu wird intern PyQt5 verwendet! Aber wir haben gar kein Fenster erstellt.



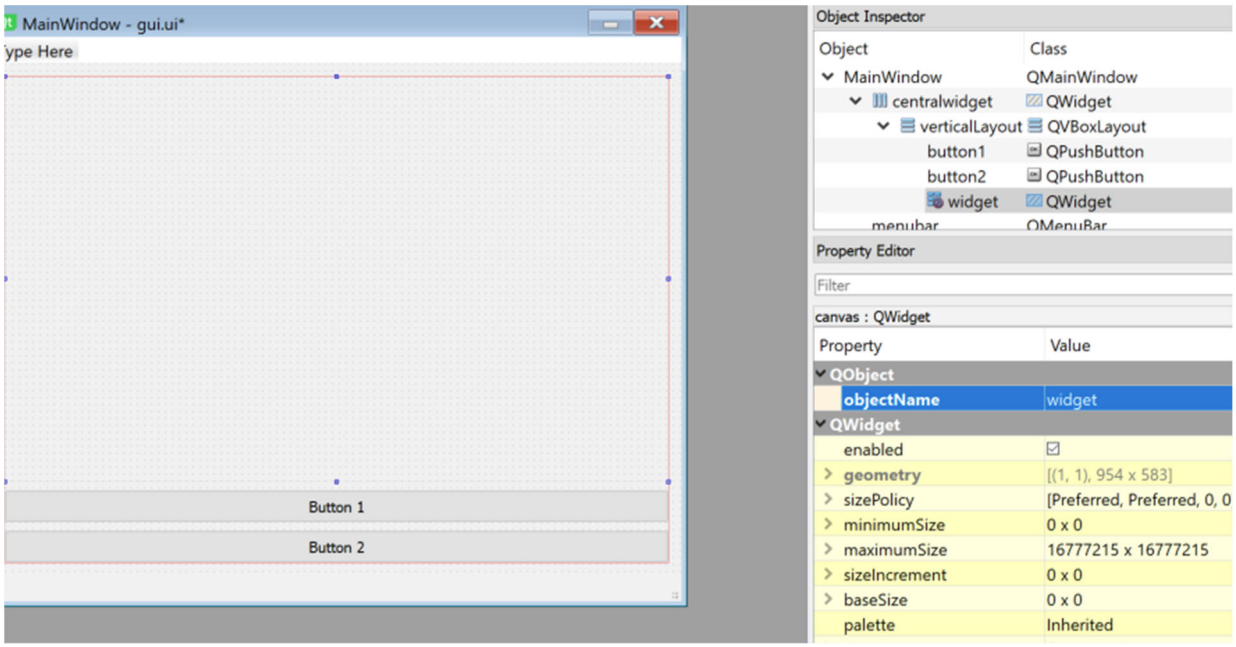
9.2 Matplotlib und PyQt5

Über das backend qt5agg kann mit PyQt5 gerendert werden («qt5 with antigrain», AGG ist eine platformunabhängige Grafikbibliothek für subpixelgenaues Rendering)

```
from PyQt5.QtWidgets import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
import matplotlib.pyplot as plt
import numpy as np


class Window(QMainWindow):
    def __init__(self):
        super().__init__()

        layout = QVBoxLayout()

        figure = plt.figure(figsize=(16,9))
        self.canvas = FigureCanvas(figure)
        #self.addToolBar(NavigationToolbar(canvas, self))

        button1 = QPushButton("y=sin(x)")
        button2 = QPushButton("y=cos(x)")

        button1.clicked.connect(self.plot1)
        button2.clicked.connect(self.plot2)

        layout.addWidget(self.canvas)
        layout.addWidget(button1)
        layout.addWidget(button2)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

    def plot1(self):
        plt.clf() # clear figure: löscht die aktuelle Figure
        x = np.linspace(-np.pi, np.pi, 20)
        y = np.sin(x)
        plt.plot(x, y, "ro-")
        plt.axis("equal")

        self.canvas.draw()
```

```
    def plot2(self):
        plt.clf() # clear figure
        x = np.linspace(-np.pi, np.pi, 20)
        y = np.cos(x)
        plt.plot(x, y, "ko-")
        plt.axis("equal")

        self.canvas.draw()


app = QApplication([])
window = Window()
app.exec()
```

Selbstverständlich kann auch der Qt-Designer verwendet werden, dazu wird einfach ein neutrales QWidget eingesetzt und später im Code ersetzt. Man könnte man es einfach auch weglassen.




```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.uic import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
import matplotlib.pyplot as plt
import numpy as np
```

```
class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("gui.ui", self)
        self.show()

        figure = plt.figure(figsize=(16, 9))
        self.canvas = FigureCanvas(figure)
        self.verticalLayout.removeWidget(self.widget)
        self.verticalLayout.insertWidget(0,self.canvas)

        self.button1.clicked.connect(self.button1click)
        self.button2.clicked.connect(self.button2click)

    def button1click(self):
        plt.clf()
        x = np.linspace(0, np.pi, 20)
        y = np.sin(x)
        plt.plot(x, y, "ro-")
        plt.axis("equal")

        self.canvas.draw()

    def button2click(self):
        plt.clf()
        x = np.linspace(0, np.pi, 20)
        y = np.cos(x)
        plt.plot(x, y, "ko-")
        plt.axis("equal")

        self.canvas.draw()

app = QApplication([])
window = MyWindow()
app.exec()
```

Wir können auch noch einen horizontalen Slider hinzufügen und den Wertebereich 10 bis 100 setzen (minimum, maximum). Der Objektname sei «slider»

```
        self.slider.valueChanged.connect(self.slidermoved)

    def slidermoved(self, value):
        plt.clf()
        x = np.linspace(0, np.pi, value)
        y = np.cos(x)
        plt.plot(x, y, "bo-")
        plt.axis("equal")
        self.canvas.draw()
```

9.3 Daten mit QLineEdit, QSlider oder QRadioButton einlesen

```
from PyQt5.QtWidgets import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
import matplotlib.pyplot as plt
import numpy as np

class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()

        figure = plt.figure(figsize=(16,9))
        self.canvas = FigureCanvas(figure)

        self.x = QLineEdit("np.linspace(0, np.pi, 20)")
        self.y = QLineEdit("np.sin(x)")
        self.button = QPushButton("Plot")

        self.button.clicked.connect(self.plot)

        layout.addWidget(self.canvas)
        layout.addWidget(self.x)
        layout.addWidget(self.y)
        layout.addWidget(self.button)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

    def plot(self):
        plt.clf()
        try:
            x = eval(self.x.text())
            y = eval(self.y.text())
        except:
            QMessageBox.error(self, "Error", "Please specify x and y")

        plt.plot(x, y, "ro-")
        plt.axis("equal")

        self.canvas.draw()

app = QApplication([])
window = Window()
app.exec()
```