05_Mandelbrot

0.1 Ein "Schmankerl" zum Schluss: Die Mandelbrotmenge

Zum Abschluss gibt es noch etwas zum Spielen: Die Mandelbrotmenge ist ein Beispiel für mathematische Kunst. Sie ist definiert als die Menge der Werte $c \in \mathbb{C}$ in der komplexen Ebene, für die die Folge

$$z_{n+1} = z_n^2 + c$$

betragsmäßig beschränkt bleibt für $z_0 = 0$ und $n \to \infty$. Beispielsweise ist $c = 1$ kein Element der Mandelbrotmenge, weil die Folge divergiert; die ersten Folgenglieder lauten $0, 1, 2, 5, 26, \ldots$. Für $c = -1$ sind die Folgenglieder $0, -1, 0, -1, 0, \ldots$, was offenbar beschränkt ist, also ist $c = -1$ Element der Mandelbrotmenge.

Wenn wir dies für hinreichend viele $c \in \mathbb{C}$ ausprobieren, und jeden Punkt c entsprechend der "Konvergenzgeschwindigkeit" der Folge einfärben, erhalten wir ein sogenanntes Fraktal, eine selbstähnliche Menge.

Die folgende Klasse kapselt einen matplotlib-plot, und enthält als besonderes Bonbon die Möglichkeit, durch das Aufziehen eines Rechtecks mit der Maus in den Plot "hineinzuzoomen", um die Selbstähnlichkeit zu verfolgen und viele weitere schöne Bildchen zu generieren.

[1]:
```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np

class MandelbrotPlot():
    """
    MandelbrotPlot gibt die Mandelbrotmenge in der komplexen
    Ebene aus. Die Mandelbrotmenge ist die Menge aller komplexen
    Zahlen c, für die die rekursiv definierte Folge z0, z1, ...
    mit dem Bildungsgesetz

    z_n+1 = z_n^2 + c, z_0 = 0

    beschränkt bleibt.
    """
    def __init__(self):
        # Start eines neuen matplotlib-Plots
```

```
        self.fig = plt.figure()
        # später benötigen wir Zugriff auf eine tiefere Ebene von matplotlib,
        # das ist neu und sollte in der Dokumentation nachgelesen werden
        self.ax = self.fig.add_subplot(111)
        # ab hier werden nur Attribute definiert und auf Standardwerte gesetzt
        self.xmin = -2.5; self.xmax = 1.0;
        self.ymin = -1.5; self.ymax = 1.5;
        self.xpress = self.xmin
        self.xrelease = self.xmax
        self.ypress = self.ymin
        self.yrelease = self.ymax
        self.resolution = 600
        self.maxiters = 100

        # Interaktion: s. https://matplotlib.org/stable/users/event_handling.html
        #
        # mit canvas kommen wir auf die Ebene von matplotlib, auf der
        # der Plot "zusammengebaut" wird.
        self.fig.canvas.mpl_connect('button_press_event', self.onpress)
        self.fig.canvas.mpl_connect('button_release_event', self.onrelease)
        self.plot_fixed_resolution(self.xmin, self.xmax, self.ymin, self.ymax)

    def mandelbrot(self, X, Y):
        """
        Vektorisierte Berechnung der Mandelbrot Menge.
        """
        C = X + Y*1j
        Z = C
        # Schnellkonstruktion einer Matrix
        # das gilt übrigens für jede Zeile in der Schleife

        # divtime speichert die Iterationszahl für jeden Punkt in der
        # komplexen Ebene, dies benötigen wir später zum Einfärben
        divtime = self.maxiters + np.zeros(Z.shape, dtype=int)
        for n in range(self.maxiters):
            Z = Z**2 + C
            diverge = Z*np.conj(Z) > 2**2 # einfache Detektion von "Divergenz"
            div_now = diverge & (divtime == self.maxiters)
            divtime[div_now] = n
            Z[diverge] = 2

        return divtime

    def plot_fixed_resolution(self, x1, x2, y1, y2):
        """
        Berechnung der Mandelbrotmenge auf der Menge
        x + iy, mit x \in [x1, x2], y \in [y1, y2]
```

```
        und Ausgabe.
        """
        x = np.linspace(x1, x2, self.resolution)
        y = np.linspace(y1, y2, self.resolution)
        X, Y = np.meshgrid(x, y)
        C = self.mandelbrot(X, Y)
        self.ax.clear()
        self.ax.set_xlim(x1, x2)
        self.ax.set_ylim(y1, y2)
        # pcolormesh verwendet als C eine selbst definierte
        # Farbpalette, nämlich divtime von weiter oben
        self.ax.pcolormesh(X, Y, C, shading="auto")
        # hier wird gezeichnet, aber noch nichts ausgegeben
        self.fig.canvas.draw()

    def onpress(self, event):
        """
        Diese Funktion wird aufgerufen, sobald der Benutzer
        eine Maustaste drückt. Die Klick-Koordinaten werden
        in Attributen gespeichert.
        """
        if event.button != 1: return
        self.xpress = event.xdata
        self.ypress = event.ydata

    def onrelease(self, event):
        """
        Diese Funktion wird aufgerufen, sobald der Benutzer
        die Maustaste loslässt. Die entsprechenden Koordinaten
        werden in Attributen gespeichert, zusammen mit den
        Startkoordinaten definiert dies das Rechteck, für das
        der Plot neu generiert wird.
        """
        if event.button != 1: return
        self.xrelease = event.xdata
        self.yrelease = event.ydata
        self.xmin = min(self.xpress, self.xrelease)
        self.xmax = max(self.xpress, self.xrelease)
        self.ymin = min(self.ypress, self.yrelease)
        self.ymax = max(self.ypress, self.yrelease)
        self.plot_fixed_resolution(self.xmin, self.xmax,
                                   self.ymin, self.ymax)


##### HAUPTPROGRAMM
plot = MandelbrotPlot()
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.2 Impressum

0.2.1 Programmierkurs Python, Dominik Göddeke https://www.ians.uni-stuttgart.de, Universität Stuttgart

Version vom April 2023

Gefördert durch die Stiftung Innovation in der Hochschullehre. (alle Rechte am Logo vorbehalten)

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie.

[ ]: