

---

Übung zum Kompaktkurs  
**Einführung in die Programmierung zur Numerik mit Python**  
Sommersemester 2017 — Hausaufgabe

---

Abgabe Ihrer Lösungen als Emailanhang mit vollständigen Namen an [felix.schindler@wwu.de](mailto:felix.schindler@wwu.de) bis spätestens Donnerstag, 05. Oktober 2017, 17:00 Uhr.

**Aufgabe 1** (Mandelbrot-Menge)

Es sei für eine komplexe Zahl  $c$  die Funktion  $\Phi_c : \mathbb{C} \rightarrow \mathbb{C}$  gegeben durch

$$\Phi_c(z) := z^2 + c.$$

Die Mandelbrotmenge  $\mathbb{M}$  ist dann definiert als die Menge aller  $c \in \mathbb{C}$ , für welche die Folge  $\Phi_c^{(k)}(0)$  beschränkt bleibt, also

$$\mathbb{M} := \left\{ c \in \mathbb{C} \mid \limsup_{k \rightarrow \infty} |\Phi_c^{(k)}(0)| < \infty \right\}.$$

Man kann leicht zeigen, dass  $\Phi_c^{(k)}(0)$  beliebig groß wird, sobald  $\Phi_c^{(k)}(0)$  einen Wert von Betrag größer 2 annimmt. Daher gilt schon

$$\mathbb{M} = \left\{ c \in \mathbb{C} \mid \sup_{k \in \mathbb{N}} |\Phi_c^{(k)}(0)| \leq 2 \right\}.$$

Um  $\mathbb{M}$  grafisch darstellen zu können, approximieren wir  $\mathbb{M}$  zunächst für eine gegebene maximale Anzahl Iterationen  $N$  durch

$$\mathbb{M}_N := \left\{ c \in \mathbb{C} \mid |\Phi_c^{(k)}(0)| \leq 2 \quad \forall k \in \mathbb{N}, k \leq N \right\}.$$

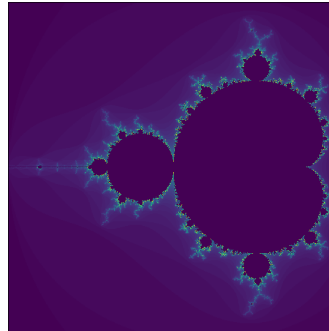
Zur Visualisierung von  $\mathbb{M}$  definieren wir nun die Funktion  $\text{mandelbrot}(x, y, N)$  als:

$$\text{mandelbrot}(x, y, N) := \begin{cases} 0 & x + iy \in \mathbb{M}_N \\ \min\{k \in \mathbb{N} \mid |\Phi_{x+iy}^{(k)}(0)| > 2\} & \text{sonst} \end{cases}, \quad (1)$$

deren Wert wir zur Färbung des Pixels zur Koordinate  $(x, y)$  verwenden.

Auf der Kurshomepage finden Sie die Datei `mandelbrot.py`, in der bereits ein Hauptprogramm implementiert ist, welches die Funktion `mandelbrot(x,y,N)` für verschiedene  $x, y$  aufruft und das Ergebnis der Auswertungen als Grafik auf dem Bildschirm darstellt. Implementieren Sie in `mandelbrot.py` die noch fehlende Funktion `mandelbrot(x,y,N)` entsprechend der Definition (1), sodass die Mandelbrotmenge korrekt dargestellt wird!

Hinweise: • Bei fehlerfreier Implementierung von `mandelbrot(x,y,N)` erhalten Sie die folgende Bildschirmausgabe:



- Sie können die Rechnungen in  $\mathbb{C}$  entweder händisch für Real- und Imaginärteil durchführen, oder den eingebauten Python-Typ `complex` verwenden.

## Aufgabe 2 (Lotka-Volterra-Modell)

Wir betrachten die Populationen  $r(t)$ ,  $b(t)$  von Räuber- und Beutetieren in einem abgeschlossenen Gehege. Wir nehmen an, dass die Beutetiere sich mit einer festen Rate  $\varepsilon_b$  vermehren, allerdings mit der Rate  $\gamma_b$  bei Zusammentreffen mit einem Räuber getötet werden. Nehmen wir an, dass die Anzahl der Treffen zwischen Räuber und Beute proportional zu  $b(t) \cdot r(t)$  mit Proportionalitätskonstante  $\mu$  ist, führt dies auf die Differentialgleichung

$$\frac{d}{dt}b(t) = \varepsilon_b \cdot b(t) - \gamma_b \cdot \mu \cdot b(t) \cdot r(t).$$

Weiter nehmen wir an, dass die Räuber mit einer festen Rate  $\varepsilon_r$  sterben, wenn Sie nicht fressen, und sich mit der Rate  $\gamma_r$  vermehren, wenn Sie ein Beutetier erlegen. Damit lässt sich die Zeitentwicklung der Räuberpopulation modellieren als

$$\frac{d}{dt}r(t) = -\varepsilon_r \cdot r(t) + \gamma_r \cdot \mu \cdot b(t) \cdot r(t).$$

Gehen wir von anfänglichen Populationen  $r_0$  und  $b_0$  von Räubern und Beutetieren aus, erhalten wir auf dem Zeitintervall  $[0, T]$  insgesamt das folgende Anfangswertproblem:

$$\begin{aligned} \frac{d}{dt}b(t) &= \varepsilon_b \cdot b(t) - \gamma_b \cdot \mu \cdot b(t) \cdot r(t) \quad \forall t \in [0, T], & b(0) &= b_0, \\ \frac{d}{dt}r(t) &= -\varepsilon_r \cdot r(t) + \gamma_r \cdot \mu \cdot b(t) \cdot r(t) \quad \forall t \in [0, T], & r(0) &= r_0. \end{aligned}$$

Approximieren wir nun die Zeitableitungen durch Vorwärts-Differenzenquotienten

$$\frac{d}{dt}b(t) \approx \frac{b(t + \Delta t) - b(t)}{\Delta t}, \quad \frac{d}{dt}r(t) \approx \frac{r(t + \Delta t) - r(t)}{\Delta t},$$

definieren  $\Delta t := T/N$  und setzen

$$b^{(n)} := b(n \cdot \Delta t), \quad r^{(n)} := r(n \cdot \Delta t), \quad n = 0, \dots, N,$$

führt uns dies auf das Gleichungssystem

$$\begin{aligned} b^{(n+1)} &= b^{(n)} + \Delta t \cdot \left( \varepsilon_b \cdot b^{(n)} - \gamma_b \cdot \mu \cdot b^{(n)} \cdot r^{(n)} \right) \quad \forall n \in 0, \dots, N-1, \quad b^{(0)} = b_0, \\ r^{(n+1)} &= r^{(n)} + \Delta t \cdot \left( -\varepsilon_r \cdot r^{(n)} + \gamma_r \cdot \mu \cdot b^{(n)} \cdot r^{(n)} \right) \quad \forall n \in 0, \dots, N-1, \quad r^{(0)} = r_0. \end{aligned} \quad (2)$$

(a) Schreiben Sie ein Programm, welches das Gleichungssystem (2) für die Konstanten

$$\begin{array}{ccccc} \varepsilon_b = 2 & \gamma_b = 0.01 & b_0 = 300 & T = 10 & \mu = 1 \\ \varepsilon_r = 1 & \gamma_r = 0.01 & r_0 = 150 & N = 10000 & \end{array}$$

löst und die Folgen  $b^{(n)}$ ,  $r^{(n)}$  in Python-Listen **b**, **r** speichert. Plotten Sie den zeitlichen Verlauf der Räuber- und Beutetierpopulationen, z.B. mit dem folgenden `matplotlib`-Code:

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot(np.linspace(0, T, N+1), b, label='Beute')
plt.plot(np.linspace(0, T, N+1), r, label='Räuber')
plt.legend()
plt.show()
```

(b) Was passiert, wenn Sie die Anzahl der Zeitschritte  $N$  auf 100 und weiter auf 20 verringern?

(c) Berechnen Sie auch die Lösung des modifizierten Zeitschrittverfahrens

$$\begin{aligned} b^{(n+1)} &= b^{(n)} + \Delta t \cdot \left( \varepsilon_b \cdot b^{(n)} - \gamma_b \cdot \mu \cdot b^{(n)} \cdot r^{(n)} \right) \quad \forall n \in 0, \dots, N-1, \quad b^{(0)} = b_0, \\ r^{(n+1)} &= r^{(n)} + \Delta t \cdot \left( -\varepsilon_r \cdot r^{(n)} + \gamma_r \cdot \mu \cdot b^{(n+1)} \cdot r^{(n)} \right) \quad \forall n \in 0, \dots, N-1, \quad r^{(0)} = r_0. \end{aligned}$$

Wie verhält sich die Lösung für  $N = 100$  und  $N = 20$ ?