## Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner, Hannes Saffrich
Simon Dorer, Sebastian Klähn

Universität Freiburg
Institut für Informatik
Wintersemester 2024

# Übungsblatt 15

**Abgabe: Montag, 17.03.2025, 09:00 Uhr**

Dieses Übungsblatt ist nur für **Studierende der Physik** verpflichtend, die den Kurs fachfremd belegen und bei erfolgreichem Abschluss 8 ECTS erhalten.

Das Übungsblatt enthält keine Punktebewertung, sondern wird insgesamt nach dem Prinzip *pass or fail* bewertet. Falls Sie während des Bearbeitens Fragen haben, können Sie sich gerne im Chat[1] oder per Mail[2] an uns wenden.

Reichen Sie Ihre Lösung bis zum 17.03.2025 um 09:00 Uhr **per Mail** an weidner@cs.uni-freiburg.de ein.

Alle anderen Studierenden können das Übungsblatt zur Klausurvorbereitung bearbeiten. Beachten Sie, dass wir ausschließlich die Übungsblätter der Studierenden der Physik korrigieren und bewerten.

### Einführung: Mandelbrot-Menge

Die Mandelbrot-Menge $M$ ist die Menge aller komplexen Zahlen $c \in \mathbb{C}$, sodass die folgende Zahlenfolge beschränkt bleibt:

$$\begin{aligned} z_0 &= 0 \\ z_{n+1} &= z_n^2 + c \end{aligned} \tag{1}$$

Die Folge gilt dabei als *beschränkt*, wenn der Betrag jedes Elements der Folge kleiner oder gleich 2 ist, d.h. wenn für alle $n \in \mathbb{N}$ gilt, dass $|z_n| \leq 2$. In dieser Aufgabe nennen wir eine Zahl $c \in \mathbb{C}$ beschränkt, wenn die zu $c$ gehörige Folge beschränkt ist.

Zum Beispiel ist $0.5 + 0i$ nicht beschränkt, da $|z_5| = |3.1533 + 0i| = 3.1533 > 2$, wobei $z_5$ wie folgt berechnet werden kann:

$$\begin{aligned} z_0 &= 0 \\ z_1 &= z_0^2 + c = 0^2 + 0.5 = 0.5 \\ z_2 &= z_1^2 + c = 0.5^2 + 0.5 = 0.75 \\ z_3 &= z_2^2 + c = 0.75^2 + 0.5 = 1.0625 \\ z_4 &= z_3^2 + c = 1.0625^2 + 0.5 = 1.6289 \\ z_5 &= z_4^2 + c = 1.6289^2 + 0.5 = 3.1533 \end{aligned}$$

Ob ein $c \in \mathbb{C}$ beschränkt ist, kann im Allgemeinen nicht in endlicher Zeit berechnet werden: Wenn überprüft wurde, dass für die ersten $n$ Zahlen der Folge der Betrag immer $|z_n| \leq 2$ ist, könnte trotzdem $|z_{n+1}| > 2$ sein - unabhängig davon wie groß $n$ gewählt wurde.

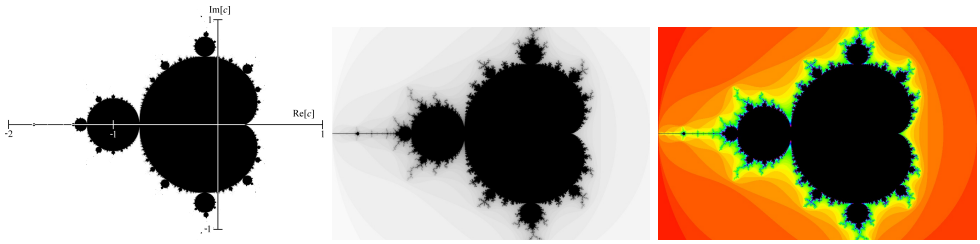
[1] Wir haben einen separaten Channel für dieses Blatt erstellt.
[2] weidner@cs.uni-freiburg.de

Die Beschränktheit kann aber näherungsweise bestimmt werden - ähnlich dem Newton-Verfahren. Hierzu wird eine maximale Zahl $m$ festgelegt und überprüft, ob die ersten $m$ Zahlen der Folge dem Betrage nach $\leq 2$ sind. Ist dies der Fall, so wird angenommen, dass dies auch für den Rest der Folge gilt und die Folge daher beschränkt ist. Je größer $m$ gewählt wird, desto höher ist der Rechenaufwand, aber auch die Chance unbeschränkte Folgen als solche zu erkennen.

Berechnet man mit diesem Verfahren, welche Zahlen $c = x + yi \in \mathbb{C}$ zur Mandelbrot-Menge gehören und zeichnet diese Punkte $(x, y)$ auf der Ebene als schwarze Punkte ein, so ergibt sich ein Bild wie in Abbildung 1a.

Für die komplexen Zahlen, die nicht beschränkt sind, kann das kleinste $n$ mit $|z_n| > 2$ bestimmt werden. Färbt man die nicht-beschränkten Zahlen so ein, dass die Helligkeit von $n$ abhängt, so ergibt sich ein Bild wie in Abbildung 1b.

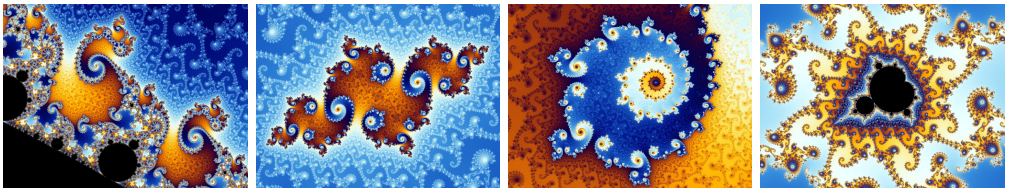
Wird nicht die Helligkeit von $n$ abhängig gemacht, sondern je nach $n$ ein anderer Farbton ausgewählt, so ergibt sich ein Bild wie in Abbildung 1c.




(a) die beschränkten Zahlen in schwarz (b) mit Einfärbung der nicht-beschränkten Zahlen (c) mit Gradienten für die nicht-beschränkten Zahlen

Abbildung 1: Visualisierung der Mandelbrot-Menge

Das Tolle an der Mandelbrotmenge ist, dass man an jede Stelle beliebig nahe heranzoomen kann. Insbesondere am Rand der Mandelbrotmenge passieren dabei sehr interessante Dinge: Da die Folgen $z_n$ chaotisches Verhalten aufweisen, entstehen dort immer wieder neue unvorhersehbare Muster, wie man z.B. in Abbildung 2 sieht.



(a) Zoom 1 (b) Zoom 2 (c) Zoom 3 (d) Zoom 4

Abbildung 2: Variationen in der Mandelbrot-Menge

Aber ein Video[3] illustriert das viel besser als ein paar einzelne Bilder. In diesem Übungsblatt sollen Sie ein Programm schreiben, welches ein Bild wie in den Abbildung 1c erzeugt.

[3] https://www.youtube.com/watch?v=u1pwtSBTnPU

**Aufgabe 15.1** (Mandelbrot: Iterationen; Datei: `mandelbrot.py`)

In dieser Aufgabe analysieren und untersuchen wir das Verhalten der Folge $z_n$ für verschiedene Werte von $c \in \mathbb{C}$.

(a) Schreiben Sie einen Generator `mandelbrot`, der eine komplexe Zahl `c` als Argument nimmt und die Werte der Folge $z_0, z_1, z_2, \ldots$ aus Gleichung 1 generiert:

Beispiel:

```
>>> g = mandelbrot(0.5 + 0.5j)
>>> [next(g) for _ in range(6)]
[0, (0.5+0.5j), (0.5+1j), (-0.25+1.5j), (-1.6875-0.25j),
↪  (3.28515625+1.34375j)]
```

(b) Schreiben Sie eine Funktion `iter_to_diverge`, die eine komplexe Zahl `c` und eine ganze Zahl `max_iter` als Argumente nimmt und das kleinste $0 \leq n \leq$ `max_iter` zurückgibt, sodass $|z_n| \geq 2$ ist. Gibt es kein solches $n$, so soll `None` zurückgegeben werden. Verwenden Sie dazu den Generator `mandelbrot`.

Beispiele:

```
>>> iter_to_diverge(0.5 + 0.5j, 50)
5
>>> iter_to_diverge(0.5 + 0.5j, 4)  # None
```

(c) Für die spätere Visualisierung wollen wir die Punkte eines Kreises berechnen. Schreiben Sie dazu eine Funktion `circle_points`, die einen Radius `r` und eine Anzahl an Punkten `num_points` als Argumente nimmt und eine Liste der Punkte eines Kreises mit Mittelpunkt $(0, 0)$ und Radius `r` zurückgibt. Die Kreispunkte sollen dabei gegen den Uhrzeigersinn durchlaufen werden.

Beispiel:

```
>>> circle_points(2.0, 4)  # 0°, 90°, 180°, 270°
[(2.0, 0.0), (1.2246467991473532e-16, 2.0), (-2.0,
↪  2.4492935982947064e-16), (-3.6739403974420594e-16, -2.0)]
>>> circle_points(8.8, 6)  # 0°, 60°, 120°, 180°, 240°, 300°
[(8.8, 0.0), (4.400000000000001, 7.62102355330306),
↪  (-4.399999999999999, 7.621023553303061), (-8.8,
↪  1.077689183249671e-15), (-4.400000000000004, -7.621023553303059),
↪  (4.399999999999994, -7.621023553303064), (8.8,
↪  -9.971348459860445e-15)]
```

(d) Nun wollen wir die einzelnen Werte der Folge $z_n$ für ein beliebiges $c \in \mathbb{C}$ visualisieren. Dazu verwenden wir das Python-Modul `matplotlib.pyplot`. Installieren Sie dieses über das Terminal mit folgendem Befehl:

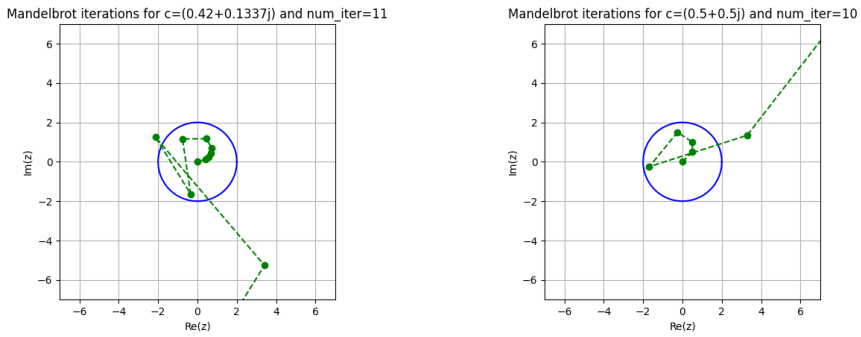
`sudo apt-get install python3-matplotlib`[4]

In der Datei `plt_intro.py` haben wir Ihnen eine kurze Einführung in die Funktionalität von `matplotlib.pyplot` bereitgestellt. Orientieren Sie sich an dieser Datei, um diese Aufgabe zu lösen.

Schreiben Sie eine Funktion `plot_iterations`, die eine komplexe Zahl `c` und eine Anzahl an Iterationen `num_iter` als Eingabe bekommt und die einzelnen Werte der

[4] Alternativ können Sie auch `python3.12 -m pip install matplotlib` verwenden. Weitere Installationshinweise finden Sie hier: https://matplotlib.org/stable/install/index.html

Folge $z_0, \ldots, z_{\texttt{num\_iter}}$ mit Hilfe von `matplotlib.pyplot` darstellt. Die komplexe Zahl `7 + 5j` wird zum Beispiel durch den Punkt mit den Koordinaten $(7, 5)$ dargestellt. Verbinden Sie die Punkte der Folge durch Linien. Zeichnen Sie außerdem die Punkte des Kreises mit Radius 2 um den Ursprung. Verwenden Sie hierzu den `mandelbrot`-Generator und die Funktion `circle_points`[5]. Ihre Plots *können* dann z.B. wie in Abbildung 3 aussehen.[6]




(a) $c = 0.42 + 0.1337i$ (b) $c = 0.5 + 0.5i$

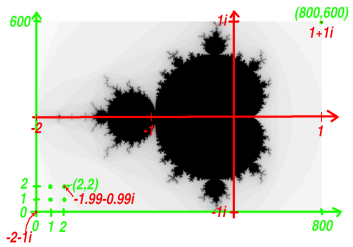
Abbildung 3: Beispiele für die Visualisierung der Folge $z_n$ mit verschiedenen Werten $c \in \mathbb{C}$.

**Aufgabe 15.2** (Mandelbrot: Bilder; Datei: `mandelbrot.py`)

In dieser Aufgabe erstellen wir Bilder der Mandelbrotmenge, wie in Abbildung 1c.

(a) Digitale Bilder bestehen aus einer Matrix von Farbpunkten - den Pixeln. Wenn wir ein Bild mit Auflösung $800 \times 600$ (800 Pixel breit, 600 Pixel hoch) berechnen wollen, so müssen wir für jedes einzelne der $800 \cdot 600 = 480000$ Pixel einen Farbwert berechnen. Jedes Pixel wird durch ein Paar von ganzzahligen Koordinaten $(x, y)$ addressiert, wobei $0 \leq x < 800$ und $0 \leq y < 600$.

Um einen Ausschnitt der Mandelbrot-Menge zu zeichnen, müssen wir zunächst die Pixelkoordinaten auf die entsprechenden komplexen Zahlen des Ausschnittes abbilden. Um diese Koordinatentransformation zu veranschaulichen, sind in der folgenden Abbildung die Pixelkoordinaten eines Bildes der Größe $800 \times 600$ in grün eingezeichnet und die zugehörigen komplexen Zahlen für den Ausschnitt $-2 - i$ bis $1 + i$ in rot:




Schreiben Sie eine Funktion

```
sample(z: complex, w: complex, x: int, y: int, sx: int, sy: int) -> complex
```

[5] Sie erhalten einen runden Kreis, indem Sie die Anzahl der Punkte groß genug (z.B. `num_points` = 100) wählen.

[6] Da `matplotlib.pyplot` kein Teil der regulären Vorlesung war erwarten wir hier kein Meisterwerk. Lassen Sie Ihrer Kreativität freien Lauf! Wenn Sie wollen können Sie natürlich beliebige weitere Funktionalitäten hinzufügen.

die für die Pixel-Koordinaten `(x,y)` eines Bildes der Auflösung `sx`×`sy` die entsprechende komplexe Zahl aus dem Ausschnitt zwischen `z` und `w` zurückgibt.

Beispiele für beliebige Gleitkommazahlen `a`, `b`, `c`, `d`:

```
sample(complex(a,b), complex(c,d), 0,   0,   800, 600) == complex(a,b)  # unten-links
sample(complex(a,b), complex(c,d), 0,   600, 800, 600) == complex(a,d)  # oben-links
sample(complex(a,b), complex(c,d), 800, 0,   800, 600) == complex(c,b)  # unten-rechts
sample(complex(a,b), complex(c,d), 800, 600, 800, 600) == complex(c,d)  # oben-rechts
sample(complex(a,b), complex(c,d), 400, 300, 800, 600) == 0.5 * complex(a,b) +
                                                          0.5 * complex(c,d)  # mittig
```

(b) Nun wollen wir der Anzahl an Iterationen, die eine Folge $z_n$ bis zur Divergenz braucht einer Farbe im HSV-Farbraum zuordnen.

Schreiben Sie eine Funktion

```
color(i: Optional[int], max_iter: int) -> tuple[int, int, int]:
```

wobei `i` die Anzahl der Iterationen bis zur Divergenz angibt (siehe Rückgabe der Funktion `iter_to_diverge`) und `max_iter` die maximale Anzahl an Iterationen, die erlaubt waren, bevor die Divergenz festgestellt wurde. Die Funktion soll ein Farb-Tupel `(hue, saturation, value)` zurückgeben. Diese werden wie folgt berechnet:

- Falls `i` `None` ist (die Folge divergiert nicht), soll die Farbe schwarz (`(0, 0, 0)`) sein.
- Der Farbton `hue` ist eine lineare, ganzzahlige Skalierung von 0 bis 255 entsprechend der Iterationszahl `i`.
- Die Sättigung `saturation` und der Wert `value` sind immer 255.

Beispiele:

```
>>> color(None, 50)
(0, 0, 0)
>>> color(0, 50)    # hue = int(0/50 * 255) = 0
(0, 255, 255)
>>> color(42, 50)   # hue = int(42/50 * 255) = 214
(214, 255, 255)
>>> color(50, 50)   # hue = int(50/50 * 255) = 255
(255, 255, 255)
```

(c) Installieren Sie die `pillow`-Bibliothek. Diese stellt Funktionalität bereit um Bilder zu Erstellen, zu Manipulieren und Abzuspeichern. Sie können die Bibliothek installieren, indem Sie auf der Kommandozeile folgenden Befehl ausführen:

```
sudo apt install python3-pil
```
[7]

Zum Testen Ihrer Installation haben wir Ihnen ein Skript `test_pillow.py` zur Verfügung gestellt. Wenn Sie dieses Skript ausführen, sollte sich ein rotes Bild `my_red_image.jpg` im Verzeichnis erstellen, in dem Sie das Skript ausgeführt haben.

Schreiben Sie eine Funktion `render_mandelbrot`, die ein Bild der Mandelbrotmenge generiert. Ruft man die Funktion wie folgt auf, so soll das Bild aus Abbildung 1c erstellt werden:

[7] Alternativ können Sie auch `python3.12 -m pip install pillow` verwenden.

```
>>> from mandelbrot import render_mandelbrot
>>> render_mandelbrot(
...         -2-1j, 1+1j,    # Intervall auf der komplexen Ebene.
...         900, 600,       # Auflösung des zu erzeugenden Bildes.
...         50,             # Anzahl maximaler Schleifendurchläufe.
...         'output.jpg')   # Dateiname des zu erzeugenden Bildes.
```

Verwenden Sie als Grundgerüst den Beispielcode aus `test_pillow.py`. Verwenden Sie außerdem die `sample`-Funktion, um die Pixelkoordinaten auf die komplexe Ebene abzubilden, um anschließend mit den Funktionen `iter_to_diverge` und `color` die Farbe des Pixels zu bestimmen.