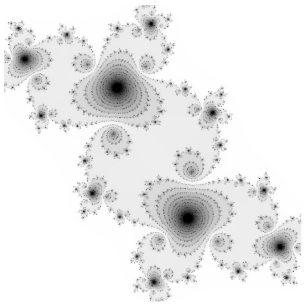

Universität Stuttgart

Mathematische Programmierung Projektarbeit 1

Aufgabe 1

Julia Fraktale

-

25.Juli.2021

Inhaltsverzeichnis

1 Vorwort

Es sei erwähnt, dass dieses Dokument mit LaTeXverfasst.
Variablen wurden in `Schreibmaschinenschrift` geschrieben und explizite Code-Zeilen zusätzlich zentriert. Die Namen von Funktionen wurde *kursiv* verfasst.
Alle verwendeten Bilder wurden mit „Julia Fraktale.py“ erzeugt und die exakten Code-Zeilen befinden sich in Kapitel 9.

2 Externe Module

Folgende Module wurden verwendet:

Name	**Abkürzung**	**Zweck**
matplotlib.pyplot	plt	Darstellung einer Matrix als Bild
matplotlib.animation	animation	Animation der Generierung
numpy	np	Für große mathematische Operationen
math	-	Für Beispiele mit π und e

Da es sich hierbei ausschließlich um Standartmodule handelt, wurden diese dem Projekt nicht beigelegt.

3 Parameter

3.1 brightness

Die Funktion *brightness* verwendet folgende Parameter:

Name	**Datentyp**	**Standartwert**	**Zweck**
`f`	np.poly1d	-	Die Funktion von der die Julia-Menge gebildet werden soll
`M`	int	500	Ergibt quadriert die Anzahl der betrachteten Punkte
`N`	int	1000	Maximale Anzahl der betrachteten Potenzen von f
`R`	float	2	Schranke, ab der aufgehört wird weitere Potenzen von f zu betrachten
`I`	list	[-1, 1]	Unter und obere Grenze des betrachteten Intervalls
`name`	str	out	Name des Bildes, was am Schluss gespeichert wird
`animation`	bool	False	Ob das Fraktal animiert werden soll
`save`	bool	False	Ob das Fraktal gespeichert werden soll

3.2 animate

Die Funktion *animate* verwendet folgende Parameter:

Name	Datentyp	Standartwert	Zweck
frame	int	-	Das Frame zu welchem das entsprechende Bild ausgegeben werden soll.

4 Features

Wie die oben beschriebenen Variablen schon angedeutet haben, besitzt die hier verwendete Visualisierungsmethode zusätzlich zu den von der Aufgabenstellung geforderten Eigenschaften noch folgende Features:

- Eine Fortschrittsanzeige
  Der prozentuale Fortschritt wird, wie in Abb. 1 gezeigt, in der Konsole ausgegeben.

- Personalisierte Namen
  Über einen String kann man den Namen des ausgegebenen Bildes frei wählen.

- Speichern als .png
  Über den Wahrheitswert der Variablen `save` lässt sich das Bild mit dem gewählten Namen direkt abspeichern.

```
Zu 67 % fertig
Zu 68 % fertig
Zu 69 % fertig
Zu 70 % fertig
Zu 71 % fertig
```

Abbildung 1

5 Dokumentation

5.1 Lokale Variablen

In diesem Abschnitt wird die Bedeutung der Verwendeten Variablen erklärt.

Die Variable `h` ist ein 2D-Array der Dimension `MxM`. In diesem Array werden die Helligkeitswerte der jeweiligen komplexen Zahl gespeichert und dementsprechend die Helligkeit dieses Pixels.

`H` ist quasi der große Bruder von `h`. Dies wird nur dann erzeugt, wenn `animate` True ist. Es ist, wie `h`, von der Dimension `MxM`. besitzt aber zudem noch eine Tiefe von `N`/10, was später bei der Initialisierung in Kapitel 5.2.1 näher erläutert wird.

In `A` läuft der eigentliche Rechen- und Iterationsprozess (Kapitel 5.2.2) ab, daher hat es ebenso die Dimension `MxM`. Hierbei ist aber wichtig, dass dieses Numpy-Array komplexe Zahlen enthält.

Die Variablen `X` und `Y` sind ähnlich aufgebaut. Beide sind Listen bestehend aus `M` Elementen, welche gleichmäßig verteile Zahlen zwischen Intervallanfangs- und Endpunkt enthalten.

`n` und `s` sind Laufvariablen, wobei erstere die momentane Iteration angibt und letztere die Schicht von `H`, was später noch erklärt wird (Kapitel 5.2.2).

`T` ist eine Liste, welche speziell ausgewählte Koordinaten von Punkten aus A enthält.

`axis` ist offensichtlicherweise die Variable, welche die Grenzen der Ebene beschreibt und somit später die Achsenbeschriftung (Kapitel 5.2.3).

Dies waren alle Variablen aus *brightness*.

Neben zuvor beschriebenen `I` und `axis` enthält *animate* die Variable S, welche eine Teilmenge von `H` und damit ein `MxM` Array ist (Kapitel 5.2.3).

5.2 Umsetzung

Wie der Name *brightness* andeutet, ist das Ziel dieser Funktion ein Array auszugeben, welches Helligkeitswerte enthält, mit welchen man ein Bild erzeugen kann. Dies wurde dann in soweit erweitert, dass nun auch die Ausgabe des Bildes Teil dieser Funktion ist.

5.2.1 Initialisierung

Für die Helligkeit gibt es 3 Fälle:

1. Fall
   Falls $|z| > \texttt{R}$ ist $h(z) = 1$
2. Fall
   Falls $|f^n(z)| > \texttt{R}$ aber für alle $k \leq n \leq \texttt{N}$ $|f^k(z)| \leq \texttt{R}$ ist $h(z) = 1 - \frac{n}{\texttt{N}}$
3. Fall
   Falls $|f^k(z)| \leq \texttt{R}$ für alle $k \leq \texttt{N}$ ist $h(z) = 0$

Der 3. Fall ist direkt durch die anfängliche Erzeugung von `h` geklärt, denn es gilt:

```
h = np.zeros(shape = (M, M))  .
```

Also h wird als Array voller Nullen erzeugt. Die Einträge werden nur verändert, falls einer der anderen Fälle eintritt.

Ähnlich wie `h` wird, wenn `animation` True ist,

```
H = np.zeros(shape = (int(N / 10) + 1, M, M))
```

erzeugt. Es sei an dieser Stelle darauf hingewiesen, dass der Animationsprozess somit nicht für `N` mod $10 \neq 0$ funktioniert.

`A` wird genau wie `h` erzeugt.

Die Listen `X` und `Y`, welche den Real- und Imaginärteil von z darstellen werden durch `np.linspace` erzeugt, was dafür sorgt, dass man die geforderten gleichmäßig verteilten Punkte erhält.

Durch die `for`-Schleife

```
for y in range(len(Y)):
    for x in range(len(X)):
        A[y, x] = complex(X[x], Y[y])
```

werden alle `x` $\in$ `X` und `y` $\in$ `Y` durchlaufen und `A` an der entsprechenden Stelle mit der komplexen Zahl z = `x` + `y` $\cdot\, i \in \mathbb{C}$ ersetzt.

An dieser Stelle wird auch der 1. Fall geprüft:

```
if np.abs(A[y, x]) > R:
    h[y, x] = 1   .
```

5.2.2 Iteration

Der nächste Teil ist quasi das Herzstück der gesamten Funktion. Hier läuft der Iterationsprozess und somit auch der noch fehlende 2. Fall ab.

Die `while`-Schleife läuft solange `n` $\leq$ `N` ist.

```
A = np.where((np.abs(A) <= R) & (np.abs(A) != R**2 + 10), f(A), A)
```

Dies durchsucht das Array nach Einträgen a, welche $|a| \leq$ `R` und $|a| \neq$ `R`$^2 + 10$ erfüllen. An einer solchen Stelle wird der Eintrag von `A` durch `f(A)` ersetzt. Ansonsten bleibt er unverändert.

$\texttt{R}^2 + 10$ wurde so gewählt, damit ein Wert konstruiert wird, welcher nie „aus versehen“ angenommen wird.

Die nächste Zeile beschäftigt sich mit dem 2. Fall:

```
T = list(zip(*np.where((np.abs(A) > R) & (A != R**2 + 10))))
```

Diese sehr komplizierte Schreibweise erzeugt eine Liste `T`, welche die Koordinaten der Einträge a aus `A` enthält, welche $|a| >$ `R` und $|a| \neq \texttt{R}^2 + 10$ erfüllen, was eben genau der 2. Fall ist.

Die Elemente dieser Liste werden mit einer `for`-Schleife durchlaufen und manipulieren `h` und `A` an den entsprechenden Stellen so, dass `h` an dieser Stelle gleich $1 - \texttt{n}/\texttt{N}$ und `A` auf den Sperrwert $\texttt{R}^2 + 10$ gesetzt wird.

Der nächste Teil beschäftigt sich mit dem Fall das `N` mod $10 = 0$ ist. Falls `animation` == True wird

```
H[s] = h  ,
```

also wird `H` alle 10 Iterationen in der Schicht `s` auf den momentanen Stand von `h` gesetzt. Dies sind dann später die `frames` der Animation (Kapitel 5.2.3). Anschließend wird `s` um 1 erhöht.

Des Weiteren wird im Fall `N` mod $10 = 0$ der prozentuale Fortschritt über den `print`-Befehl ausgegeben.

Damit die `while`-Schleife irgenwann terminiert wird schlussendlich `n` um eins erhöht.

5.2.3 Vom Array zum Bild

Der nun folgenden Teil beschäftigt sich mit der Darstellung eines Array durch `matplotlib`.

```
plt.imshow(h, cmap = "gray", vmin = 0, vmax = 1, extent = axis)
```

spielt hierbei die zentrale Rolle.
Als „Farbe“ wird hier `gray`, also Graustufen verwendet, sodass für den Arrayeintrag `0` Schwarz und für `1` Weiß erzeugt wird. Schlussendlich wird mit `extent` die Achsenbeschriftung erzeugt.

Nun wird noch im Falle, dass `save` gewünscht ist, der `plot` als `name`.png gespeichert und mit `plt.show()` angezeigt.

Die Ausgabe von *brightness* hängt davon ab, ob `animation` gefordert ist. In diesem Fall wird `H` zurückgegeben und ansonsten `h`.

5.2.4 Animation

Die Funktion *animate* erzeugt basierend auf der Nummer des frames und somit der framten Schicht von `H`, welche mit `S` bezeichnet ist, über `im.show(...)` wie oben das Bild `im`, was schlussendlich auch die Ausgabe von *animate* ist.

5.2.5 Polynome

Das Ende des Codes bilden einige Aufrufe von *brightness* zu verschiedenen Polynomen `f`, `R` und `N`, welche in Kapitel 7 genauer erläutert werden.

```
animation.FuncAnimation(fig, animate, frames = 100, interval = 200,
    blit = True)
```

erzeugt die in (d) geforderte Animation mit `matplotlib.animation`. Mit insgesamt 100 Bildern und 0,2 Sekunden Pause zwischen den Bilder, also 5 fps.

6 Benutzungshinweise

Dank der Ausgabe des Fortschritts lässt sich leicht nachvollziehen, ob das Programm läuft. Die Berechnung dauert auf meinem PC[1] für alle Polynome jeweils unter 20 Sekunden.

Eigene Julia-Fraktale lassen sich über den Aufruf der *brightness*-Funktion relativ einfach erzeugen. Es wird nur ein Polynom benötigt. Wenn man ein „einfaches“ Julia-Fraktal erstellen möchte, braucht man einen Ausdruck der Form $\texttt{f}(z) = z^2 + c$ mit $c \in \mathbb{C} \setminus \mathbb{R}$.[2] Demnach ist bei `np.poly1d([1,0, complex(a, b)])` noch die (komplexe) Konstante $\texttt{a} + \texttt{b} \cdot i$ zu wählen.

Man kann nach belieben mit den Variablen `M`, `N`, `R` und `I` spielen. Es sei aber zu beachten, dass der Rechenaufwand für `M` quadratisch und für `N` linear ansteigt.

Wenn man sein wunderschönes Fraktal gerne als .png haben wöllte, muss die Funktion mit `save = True` übergeben werden.

Wer eine Animation seines Polynoms sehen will, muss das Polynom in

[1] Intel i7-6700K, 32GB RAM, NVIDIA GTX 980 Ti
[2] mit reellen c ist es langweilig

```
H = brightness(f = np.poly1d([1, 0, complex(-0.1, 0.651)]), name =
    "Atoll", animation = True, save = True)
```

ändern.
Falls man ein anderes `N` möchte, funktioniert die Animation nur, wenn man in `animation.FuncAnimation` die `frames` auf `N`/10 setzt. Veränderte Intervallgrenzen müssen *brightness* übergeben werden und in *animate* entsprechend verändert werden.

7 Bilder und Beispiele

Die Fraktale erhielten Name je nachdem wonach sie aussahen; diese werden hier als Überschriften verwendet und die exakten Einstellungen, die zu den Bilder führten werden im Anhang (Kapitel 9) beschrieben.

7.1 Atoll

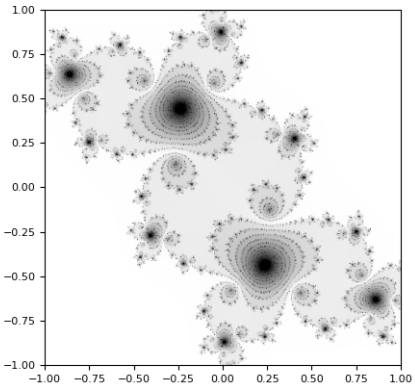

Abbildung 2

Dies ist das Fraktal zu (c) also $\texttt{f}(z) = z^2 - 0.1 + 0.651 \cdot i$.

7.2 Elefant

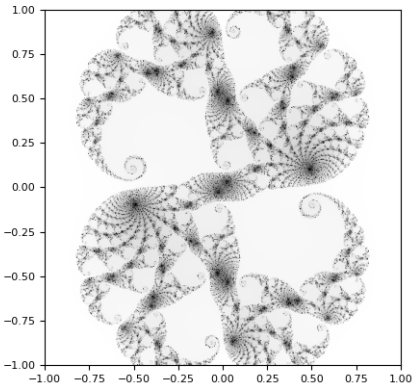

Abbildung 3

Dies ist ein Fraktal zu $\mathtt{f}(z) = z^2 + 0.26 + 0.0016 \cdot i$.

7.3 Illusion

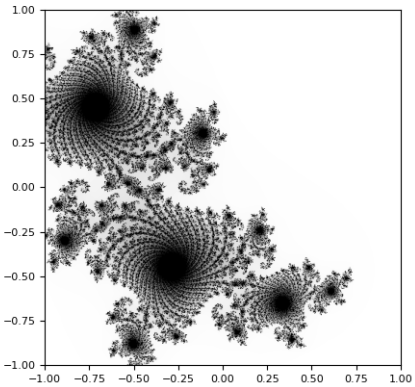

Abbildung 4

Dies ist ein Fraktal zu $\mathtt{f}(z) = z^2 + z - 0.306 + 0.648 \cdot i$ mit $\mathtt{N} = 500$.[3]

7.4 Fancy

Manche Leute behaupt $e^{i \cdot \pi} - 1 = 0$ sei die schönste Gleichung der Mathematik, da sie die wichtigen Konstanten e, π und i mit den neutralen Elementen der Multiplikation

[3] Wenn man auf die großen schwarzen Punkte schaut, scheinen diese größer zu werden. Ich kann Ihnen aber versichern, dass dies nur ein Bild ist und es demnach eine optische Illusion sein muss.

und Addition verbinden.
Wem das gefällt, wird auch folgendes Fraktal gefallen:

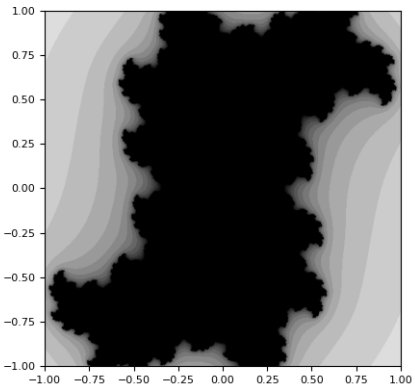

Abbildung 5

Dies ist ein Fraktal zu $\texttt{f}(z) = z^2 + 1/\pi - 1/e \cdot i$ mit $\texttt{R} = 15$ und `N = 15`.
Es verbindet also genauso diese schönen Zahlen.[4]

8 Variation der Variablen

Abschließend stellt sich noch die Frage: Wie beeinflussen die Variablen das Fraktal?[5]

8.1 `f`

Das betrachtete Polynom `f` beeinflusst grundlegend die Struktur, wie man an den oberen Beispielen leicht erkennen kann.
Eine Animation von verschiedenen Polynomen $f_c(z) = z^2 + c$ lässt sich in diesem Video[6] anschauen.

8.2 `M`

Die Variation von `M` verändert die Auflösung des Bildes:

[4]Die 0 kann man einfach gedanklich dazu addieren.
[5]Der Titel wurde in Anlehnung an die „Variation der Konstanten“ gewählt.
[6]https://youtu.be/fAsaSkmbF5s?t=788

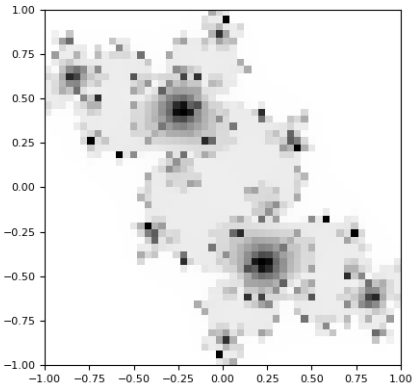

Abbildung 6

Dies ist das Fraktal zu $\texttt{f}(z) = z^2 - 0.1 + 0.651 \cdot i$ und $\texttt{M} = 50$.
Offensichtlich ist es viel gröber gepixelt als das Bild von oben.

8.3 N

Die Variation von `N` verändert die Helligkeit:

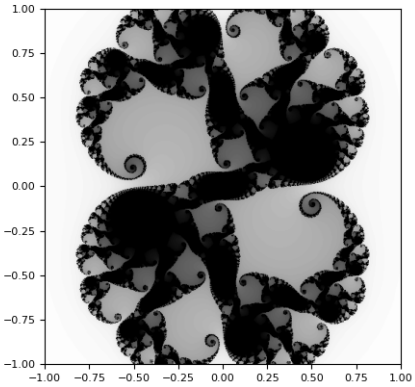

Abbildung 7

Dies ist ein Fraktal zu $\texttt{f}(z) = z^2 + 0.26 + 0.0016 \cdot i$ und $\texttt{N} = 100$.

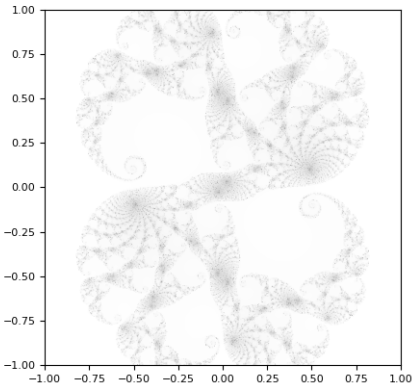

Abbildung 8

Dies ist ein Fraktal zu $\texttt{f}(z) = z^2 + 0.26 + 0.0016 \cdot i$ und $\texttt{N} = 4000$.
Also größere `N` machen das Bild heller und kleinere dunkler.

8.4 `R`

Die Variation von `R` verändert die Helligkeit minimal.

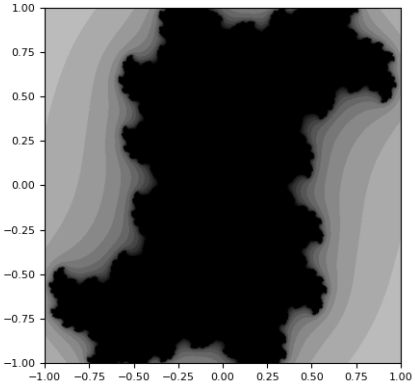

Abbildung 9

Dies ist ein Fraktal zu $\texttt{f}(z) = z^2 + 1/\pi - 1/e \cdot i$ mit $\texttt{R} = 5000$ und `N = 15`.
Für kleine `N` wird das Bild für große `R` dunkler.

8.5 `I`

Die Variation von `I` verändert den betrachteten Bereich des Bildes.

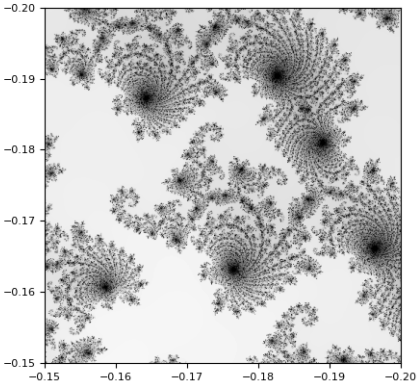

Abbildung 10

Dies ist ein Fraktal zu $\mathtt{f}(z) = z^2 + z - 0.306 + 0.648 \cdot i$ mit $\mathtt{I} = [-0.15, -0.2]$.

9 Anhang

- Abbildung 2
```
brightness(f = np.poly1d([1, 0, complex(-0.1, 0.651)]), name =
"Atoll", save = True)
```

- Abbildung 3
```
brightness(f = np.poly1d([1,0, complex(0.26, 0.0016)]), name =
"Elefant", save = True)
```

- Abbildung 4
```
brightness(f = np.poly1d([1, 1, complex(-0.306, 0.648)]), N = 500,
name = "Illusion", save = True)
```

- Abbildung 5
```
brightness(f = np.poly1d([1,0, complex(1/math.pi, -1/math.e)]), R =
15, N = 15, name = "Fancy", save = True)
```

- Abbildung 6
```
brightness(f = np.poly1d([1, 0, complex(-0.1, 0.651)]), M = 50,
name = "Atoll", save = True)
```

- Abbildung 7
```
brightness(f = np.poly1d([1,0, complex(0.26, 0.0016)]), N = 100,
name = "Elefant", save = True)
```

- Abbildung 8
```
brightness(f = np.poly1d([1,0, complex(0.26, 0.0016)]), N = 4000,
name = "Elefant", save = True)
```

- Abbildung 9
```
brightness(f = np.poly1d([1,0, complex(1/math.pi, -1/math.e)]), R =
4000, N = 15, name = "Fancy", save = True)
```

- Abbildung 10
```
brightness(f = np.poly1d([1, 1, complex(-0.306, 0.648)]), I =
[-0.15, -0.2], name = "Illusion", save = True)
```