Wissenschaftliches Rechnen

Aufgabenblatt 6 (Praxis)

wr@cg.tu-berlin.de

WiSe 2019/2020

Allgemeine Hinweise:

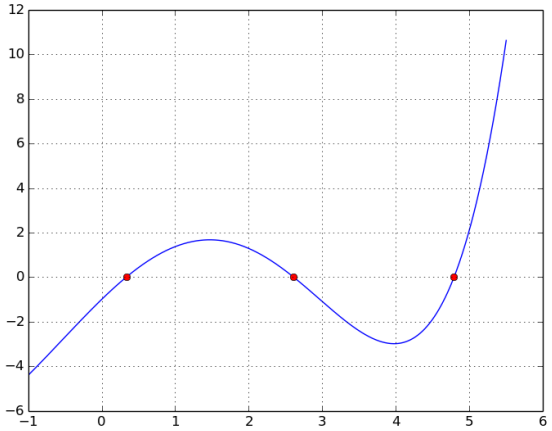
- Die Aufgaben sind von jeder/m Studierenden *einzeln* zu bearbeiten und abzugeben (Plagiate werden entsprechend der Studienordnung geahndet).
- Verwenden Sie die vorgegebene Code-Basis. Die zu implementierenden Funktionen befinden sich in der Datei **main.py**. Ihr Code ist an den mit `# TODO: ...` gekennzeichneten Stellen einzufügen. Die `NumPy`-Funktionen, welche Sie zur Lösung einer Aufgabe nicht verwenden dürfen, sind unter `Forbidden` in der Docstring Beschreibung der entsprechenden Funktion aufgelistet.
- Wir stellen einige rudimentäre Unit-Tests zur Verfügung, welche Sie verwenden sollen, um die Funktionalität ihres Codes zu testen. Sie sollten diese Tests während der Implementierung Ihrer Lösung vervollständigen (Funktionalität beschrieben in Python `unittest`). Sie können die Tests mit dem Aufruf `python3 tests.py -v [Tests.test_<function>]` ausführen.
- Bitte reichen Sie die Datei `main.py` mit Ihren Lösungen bis **Montag, den 17.02.2020, um 14:00 Uhr** auf `https://autolab.service.tu-berlin.de` mit ihren Zugangsdaten ein. Ein mehrfacher Upload bis zum Abgabeende ist möglich. Die letzte Version wird bewertet.

Aufgabe 1: Bestimmung von Nullstellen (3 Punkte)

In dieser Aufgabe sollen Nullstellen mittels Bisektion und dem Newton-Verfahren bestimmt werden. Als Testfunktion $f : \mathbb{R} \mapsto \mathbb{R}$ betrachten wir:

$$f(x) = 0.009x^5 + 0.02x^4 - 0.32x^3 - 0.54x^2 + 3.2x - 1.0.$$

Der Graph von $f(x)$ aus dem auch die ungefähre Lage der Nullstellen hervorgeht ist nachfolgend dargestellt.



Aufgabe 1.1: Bisektionsverfahren (1 Punkt)

Implementieren Sie das Bisektionsverfahren zur Nullstellenbestimmung. Verwenden Sie hierzu die Funktion `find_root_bisection()`. Falls keine minimale Intervalgröße `ival_size` gegeben ist, soll diese geeignet gewählt werden.

Aufgabe 1.2: Newtonverfahren (2 Punkte)

Implementieren Sie das Newton-Verfahren zur Bestimmung von Nullstellen in der gegebenen Funktion `find_root_newton()`. Das Verfahren soll für reelle und komplexe Startwerte funktionieren. Wählen Sie geeignete Abbruchbedingungen für die verschiedenen Probleme, die auftreten können (siehe Kommentare).

Aufgabe 2: Newton-Fraktale (2 Punkte)

Gegeben sei eine Funktion $f : \mathbb{C} \mapsto \mathbb{C}$ mit den Nullstellen $\{z_0, z_1, \ldots\}, f(z_i) = 0$. Diese (komplexen) Nullstellen lassen sich durch das Newton-Verfahren finden, wobei der Startpunkt z des Newton-Verfahrens bestimmt, welche der Nullstellen gefunden wird. Als *Attraktor* von z_i bezeichnet man die Teilmenge der komplexen Zahlenebene $\Omega_i \subseteq \mathbb{C}$ deren Elemente als Startwerte für das Newton-Verfahren zur Nullstelle z_i führen. Die Attraktoren können selbstähnliche Strukturen bilden und werden deshalb *Newton-Fraktale* genannt. Zur Darstellung identifiziert man die Nullstellen mit verschiedenen Farben und wählt die Helligkeit nach der Anzahl der benötigten Schritte bis zur Konvergenz des Verfahrens.

Implementieren Sie die Funktion `generate_newton_fractal()`, die für eine gegebene Funktion f das *Newton-Fraktal* berechnet. Hierbei müssen Sie die bereits implementierte Funktion `find_root_newton()` aus Aufgabe 1.2 verwenden. Die Parameter von `generate_newton_fractal()` sind die Funktion (`f`) und deren Ableitung (`df`), sowie deren Nullstellen (`roots`). Zusätzlich wird ein Parameter `sampling` übergeben, welcher die endliche Menge der komplexen Startpunkte für das Newton-Verfahren enthält. Die Rückgabe der Funktion gibt für jeden Startpunkt den Index der gefundenen Nullstelle sowie die Anzahl der benötigten Iterationen an. Vergleichen Sie ihre Ergebnisse mit den mitgelieferten Bildern.

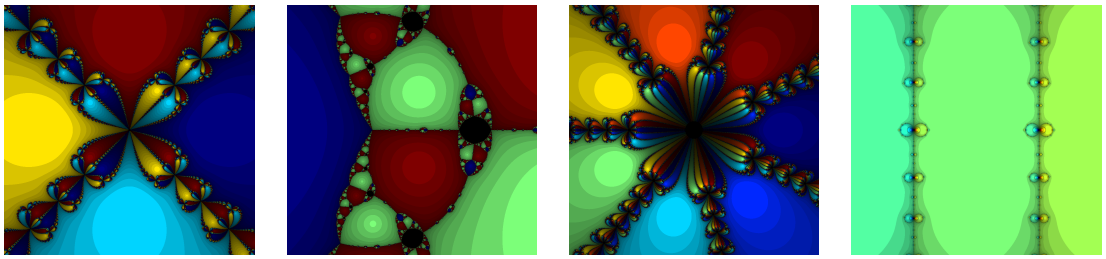


Abbildung 1: Newton-Fraktale für die Funktionen $x^4 - 1$, $x^3 - 2x + 2$, $x^7 - 1$ und $\sin(x)$.

Aufgabe 3: Minimalflächen (5 Punkte)

In dieser Aufgabe sollen Minimalflächen mithilfe des Gradientenabstiegverfahrens approximiert werden. Eine Oberfläche ist minimal, wenn sie, unter gegebenen Randbedingungen minimalen Flächeninhalt hat. Die Randbedingungen sorgen dafür, dass Teile der Fläche fixiert sind, andernfalls könnte die Fläche beliebig klein werden.

Als Oberflächen betrachten wir Dreiecksnetze. Ein Dreiecksnetz mit n_v Ecken in n_f Dreiecken wird repräsentiert durch die `NumPy` Arrays, $\mathcal{V} \in \mathbb{R}^{n_v \times 3}$ und $\mathcal{F} \in \mathbb{N}^{n_f \times 3}$. Dabei beschreibt die i-te Zeile des Arrays $\mathcal{F}$ das i-te Dreieck als Indizes in das Array $\mathcal{V}$, das die Positionen der Ecken im $\mathbb{R}^3$ enthält. Die Eckpunkte des i-ten Dreiecks sind also (in Python Notation) $\mathcal{V}[\mathcal{F}[i,0]], \mathcal{V}[\mathcal{F}[i,1]]$ und $\mathcal{V}[\mathcal{F}[i,2]]$.

Es gilt folgendes Optimierungsproblem zu lösen: Gegeben sei eine feste Dreiecksstruktur $\mathcal{F}$, initiale Positionen der Ecken $\mathcal{V}$ sowie ein `NumPy` Array der Indizes fixierter Ecken $\mathcal{C} = [c_0, c_1, c_2 ...]$, bestimme die Positionen der Ecken $\mathcal{V}'$, so dass die Gesamtfläche $A(\mathcal{V}', \mathcal{F})$ des Dreiecksnetzes minimal ist, unter der Bedingung, dass die Positionen der fixierten Ecken mit den initialen Positionen übereinstimmen: $\mathcal{V}[i] = \mathcal{V}'[i], i \in \mathcal{C}$.

Aufgabe 3.1: Flächenberechnung (1 Punkt)

Bestimmen Sie in der Funktion `surface_area()` die Gesamtfläche des Dreiecksnetzes $A(\mathcal{V}', \mathcal{F})$. Am einfachsten geht dies, indem Sie eine Funktion implementieren, die den Flächeninhalt eines einzelnen Dreiecks aus den gegebenen Eckpunktpositionen berechnet, und über alle Dreiecke (in $\mathcal{F}$) aufsummieren.

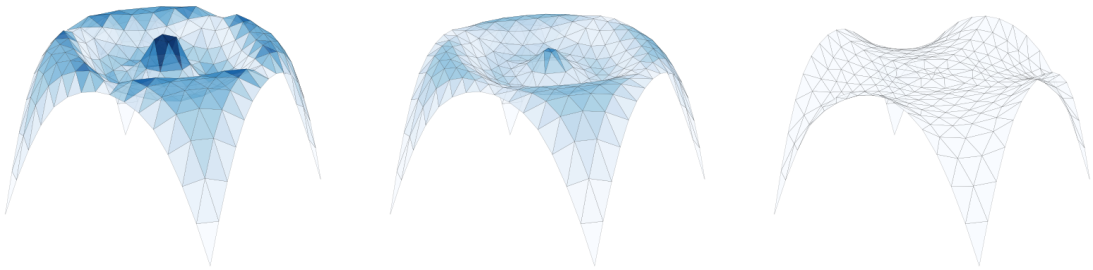


Abbildung 2: Beispiel einer Minimalfläche vor, während und nach der Optimierung. Die Farben codieren den Betrag des Gradienten der Flächen, dunkleres Blau bedeutet einen größeren Gradienten.

Aufgabe 3.2: Gradientenbestimmung (2 Punkte)

Um $\mathcal{V}'$ ausgehend von $\mathcal{V}$ mittels Gradientenabstieg zu finden, muss der Gradient von $A(\mathcal{V}', \mathcal{F})$ bezüglich der Veränderlichen gebildet werden. Die Veränderlichen sind die $3 \times n_v$ Koordinaten der Ecken. Sie könnten diese Aufgabe lösen, indem Sie den Ausdruck für den Flächeninhalt nach den einzelnen Eckpunktkoordinaten ableiten. Dies ist allerdings recht mühsam.

Einfacher geht es, wenn man Linearität der Ableitung und geometrische Anschauung verwendet. Der Gradient der gesamten Dreiecksfläche ergibt sich durch Aufsummieren der Gradienten pro Dreieck (Linearität). Ein Dreieck beeinflusst den Gradienten nur in drei Ecken. Die drei Ecken können unabhängig voneinander betrachtet werden (Eigenschaft partieller Ableitungen). Es genügt also, sich zu überlegen, was der Gradient eines Dreiecks bzgl. einer Ecke ist. Hier hilft geometrische Anschauung: der Gradient ist orthogonal zu den Richtungen, entlang derer sich der Flächeninhalt nicht ändert. Das Kreuzprodukt dieser Richtungen liefert also die Richtung des Gradienten. Die Länge des Gradienten (in einer Ecke) ist proportional zur Veränderung der Dreiecksfläche bei Bewegung der Ecke entlang des Gradienten und ergibt sich ebenfalls aus elementaren geometrischen Überlegungen.

Implementieren sie die Berechnung des Gradienten in der Funktion `surface_area_gradient()`.

Aufgabe 3.3: Gradientenabstieg (2 Punkte)

Implementieren sie die Funktion `gradient_descent_step()` die einen Schritt des Gradientenabstiegverfahrens ausführt. Bestimmen sie dazu den Gradienten und eine Schrittweite, die so gewählt werden muss, dass die Flächeninhaltsfunktion $A(\mathcal{V}', \mathcal{F})$ kleiner wird. Fangen Sie dazu mit einer großen Schrittweite an und verkleinern Sie den Wert, bis die Bedingung erfüllt ist. Gibt es keine solche Schrittweite, so war das Verfahren erfolglos. Wählen Sie ein geeignetes Kriterium zum Abbruch des Verfahrens bei Erfolg (Konvergenz) und Misserfolg.