2. Semesterarbeit - Grafiken mit Matplotlib & Python

April 24, 2020

Autor	Patrick Michel
6. Semester	FFHS - Fernfachhochschule Schweiz
Fach	AnPy, Analysis mit Python, BSc INF 2017 Pas, BE1-I, FS20
Dozent	Geuss Markus

1 Inhaltsverzeichnis

2 Einführung

In dieser Arbeit geht es um das numerische Integrieren und somit Bestimmten von Flächen einer Funktion. Dies wird in einem theoretischen Teil beschrieben sowie exemplarisch mit der Programmiersprache Python umgesetzt.

2.1 Aufgabenstellung

1. Beschreiben Sie den Begriff des bestimmten Integrales.
2. Beschreiben Sie die Sekantenzrapezregel, die Tangententrapezregel und die Simpsonsche Regel als Näherungsformel für zur Berechnung bestimmter Integrale.
3. Erstellen Sie drei Python-Funktionen
   - `sekanten_trapez_regel(f, a, b, n)`
   - `tangenten_trapez_regel(f, a, b, n)`

- `simpson_regel(f, a, b, n)`
- zur Approximation des Integrals

$$\int_a^b f(x)\,dx \tag{1}$$

- dabei steht n für die Anzahl Streifen (bzw Doppelstreifrn bei Simpson) in den Approximationen.

4. Verwenden Sie die drei Approximationsregeln zur Bestimmung der folgenden Integrale (Variieren Sie dabei die Anzahl Streifen n):

$$\int_0^\pi sin(x)\,dx \tag{2}$$

$$\int_{-1}^1 \sqrt{1-x^2}\,dx \tag{3}$$

$$\int_0^1 x^2\,dx \tag{4}$$

5. Stellen Sie Vermutungen auf, was beim Grenzwert n→∞ passiert. 6. Überprüfen Sie Ihre Vermutungen mit Sympy. 7. Vergleichen Sie die Genauigkeit der gemachten Approximationen.

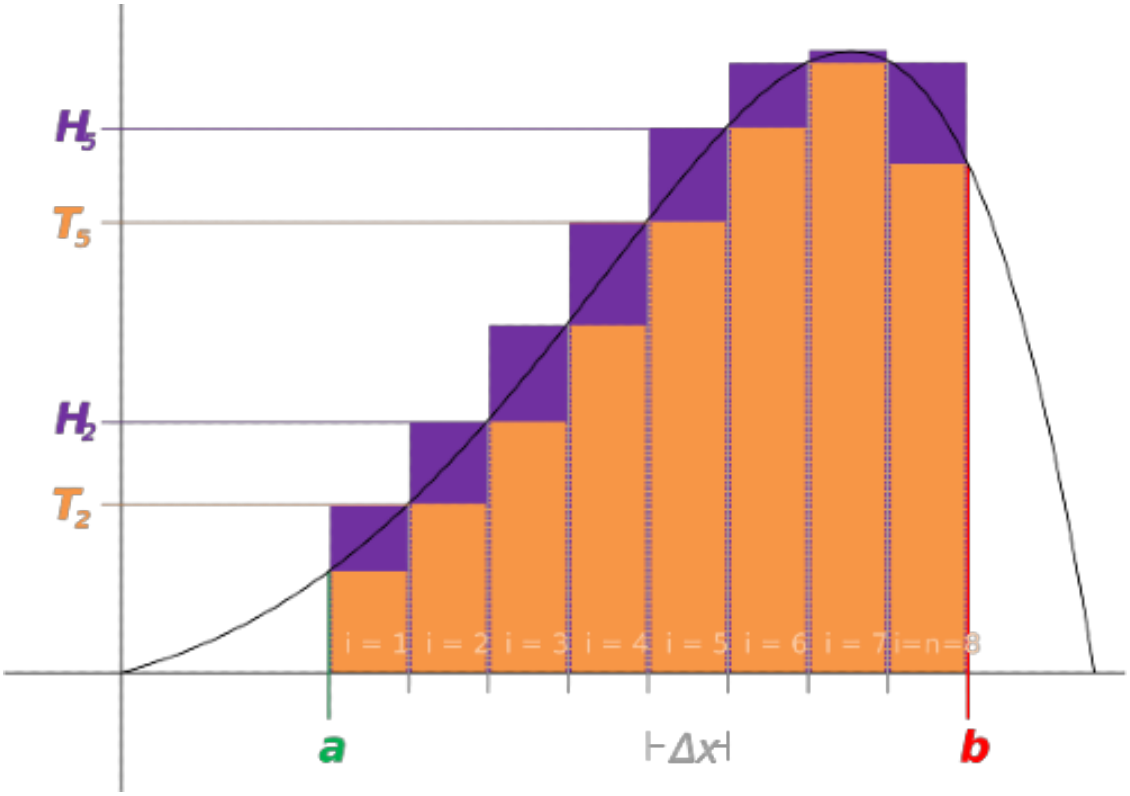
2.2 Systemanforderungen

Folgende Voraussetzungen muss die Jupyter Umgebung erfüllen um alle nachfolgenden Codebeispiele ausführen zu können.

- Python 3.7.x
- Pip Packages
  - Matplotlib
  - Numpy
  - SciPy

3 Riemann-Integral

Das Riemann Integral ist eine mathematische Methode zur Veranschaulichung des Flächeninhaltes zwischen dem Abschnitt einer x-Achse und dem Graphen einer Funktion.



Das Grundprinzip beruht auf dem Prinzip der Annäherung. Es wird zuerst ein Bereich auf der x-Achse den es zu integrieren gibt definiert. Im obigen Beispiel ist dieser Bereich von a -> b. Danach werden n Punkte welche in einem gleichen Abstand auf dem Funktionsverlauf liegen genommen und aus diesen dann gleich grosse Rechtecke erstellt. Daraus ergibt sich nun eine mehr oder weniger genau Abdeckung der Fläche dieser Funktion. Um die Fläche nun zu bestimmen kann man die Flächen der einzelnen Rechtecke summieren. Je grösser nun n wird also je kleiner die einzelnen Rechtecke werden desto genauer wird das Resultat in Bezug auf die tatsächliche Fläche. Wenn mann n nun an ∞ also den Grenzwert annähert kommt schlussendlich der exakte Wert der Fläche heraus. Doch je grösser n ist desto aufwändiger wird natürlich auch die Berechnung für den Computer. Ein Integral von hand zu bestimmen und berechnen ist theoretisch möglich doch macht durch den schnell wachsenden Aufwand keinen Sinn. Daher kann man das gut den Computern überlassen.

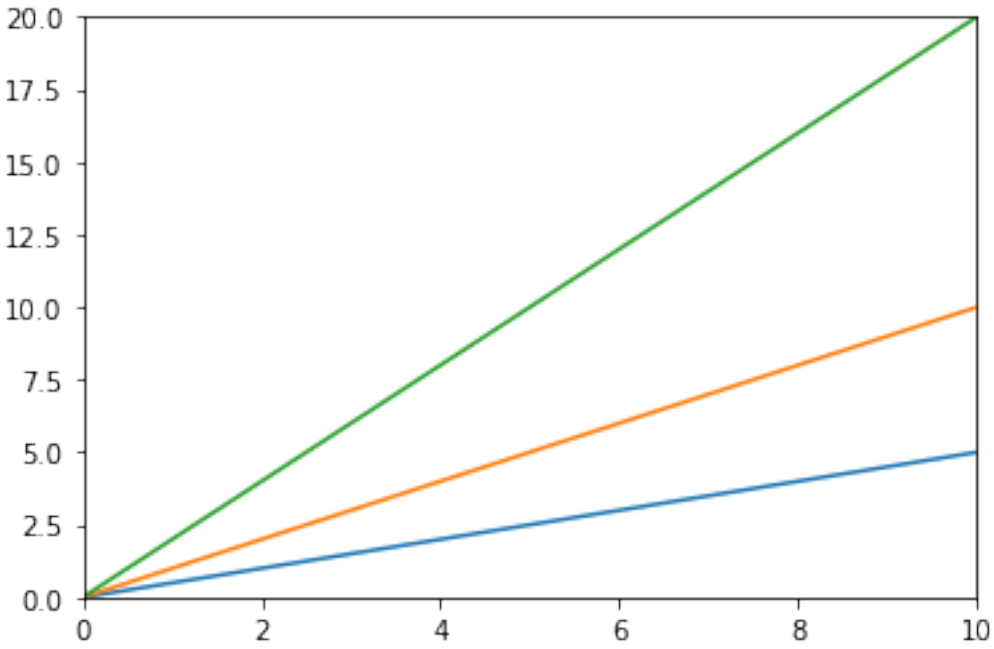
[9]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.special
import requests
import json
import networkx as nx
```

Um nachfolgend die Übersicht zu wahren wurden alle nötigen imports in die erste Sektion geholt. Für eine einfachere Handhabung wird der importierte pyplot der alias *plt* vergeben. Nachfolgend soll die grundlegende Funktionsweise anhand eines einfachen Beispiels verdeutlicht werden [2].

[10]:
```
x = np.arange(0, 11)

# flasche lineare Steigung
plt.plot(x, x * 0.5)
# normale lineare Steigung
plt.plot(x, x)
# spitze lineare Steigung
plt.plot(x, x * 2)

plt.axis([0, 10, 0, 20])
plt.show()
```



3.1 Funktionsgraphen

Funktionsgraphen sind eine graphische Veranschaulichung von Funktionen und deren Input sowei Output. In der Informatik wird bei der Algorithmik viel von den sogenannten Laufzeitklassen gesprochen. Nachfolgend einige der gängigsten Laufzeitklassen aufsteigen sortiert nach der Laufzeit[3]:

$$\mathcal{O}(1) \tag{5}$$

Die Laufzeit ist immer 1

$$\mathcal{O}(log(n)) \tag{6}$$

Die Laufzeit ist der Logarithmus von n

$$\mathcal{O}(n) \tag{7}$$

Die Laufzeit ist linear n

$$\mathcal{O}(n^2) \tag{8}$$

Die Laufzeit ist quadratisch n

$$\mathcal{O}(n!) \tag{9}$$

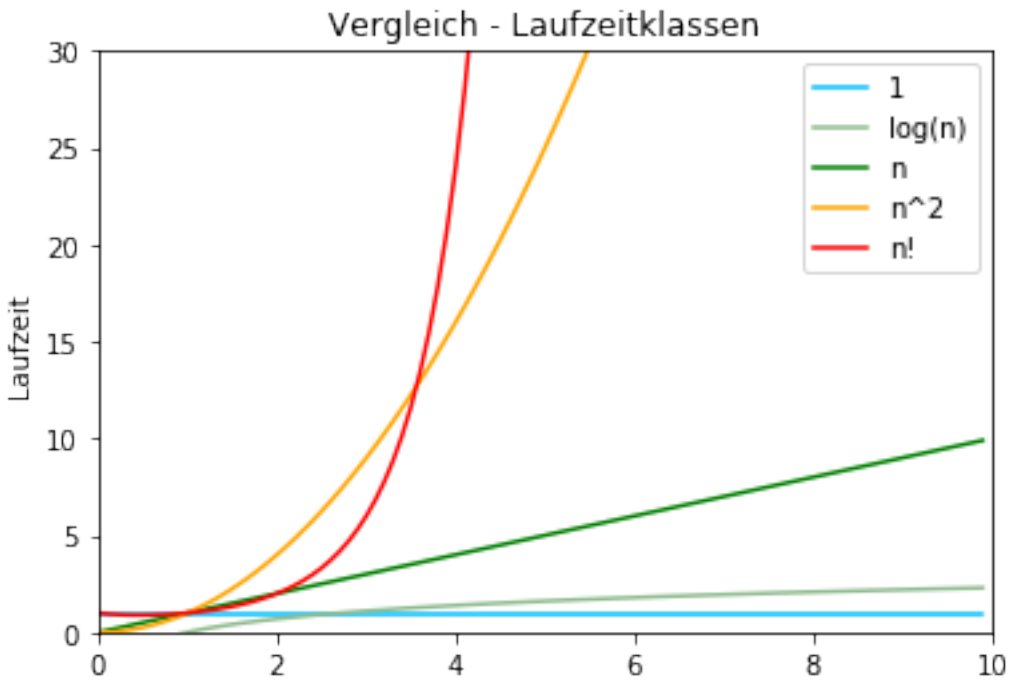
Die Laufzeit ist Fakultät n

Um diese Graphen und ihr unterschiedliches Wachstum zu verdeutlichen soll folgender Graph dienen.

[11]:
```
# Erzeuge x Werte von 1 bis 10 in 0.1 Schritten für flachere Verlaufskurven
x = np.arange(0, 10, 0.1)
# Den Graphen und die Y-Achse beschriften
plt.title('Vergleich - Laufzeitklassen')
plt.ylabel('Laufzeit')
plt.axis([0, 10, 0, 30])

# Laufzeit 1
plt.plot(x, np.full(len(x), 1), label='1', color='deepskyblue')
# Laufzeit log(n)
plt.plot(x, np.ma.log(x), label='log(n)', color='darkseagreen')
# Laufzeit n
plt.plot(x, x, label='n', color='green')
# Laufzeit n^2
plt.plot(x, x * x, label='n^2', color='orange')
# Laufzeit n!
plt.plot(x, scipy.special.factorial(x), label='n!', color='red')

plt.legend()
plt.show()
```



3.2 Reflexion & Einschätzung

Wie oben gezeigt bieten die vorgestellten Pakete oder Bibliotheken extrem viele Möglichkeiten Daten zu visualisieren. Nicht alle Darstellungsformen machen für alle Daten Sinn daher ist es wichtig die Darstellungsform immer auf die Daten und die daraus zu gewinnenden Daten abzustimmen. Matplotlib ist in Kombination mit Numpy extrem effizient und sehr einfach zu handhaben. Meiner persönlichen Meinung dienen diese Pakete in Kombination als eine Alternative für die Programmiersprache R welche sehr ähnliche Möglichkeiten bietet. Auch ist es sehr wertvoll bestimmte Themen noch einmal programmatisch zu visualisieren. Dazu zählen auch die Laufzeitklassen welche durch das in einem oberen Kapitel erstellten Diagramm extrem gut sichtbar gemacht werden konnten. Auch bietet die Programmiersprache Python sehr viele sprachliche Besonderheiten welche es extrem einfach machen mit Listen zu arbeiten. Die Standarddarstellung der Graphen ist nicht immer extrem aussagekräftig. Daher ist es eigentlich immer nötig diese den eigenen Bedürfnissen anzupassen. Dazu gehört das Anpassen der Farben oder auch das Sortieren der Daten damit sie schlussendlich mehr Sinn ergeben.

3.3 Verweise

https://www.math.ubc.ca/~pwalls/math-python/integration/riemann-sums/

- [1]: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
- [2]: https://pythonbasics.org/matplotlib-line-chart/
- [3]: https://de.wikipedia.org/wiki/Landau-Symbole
- [4]: https://bodo-schoenfeld.de/jupyter-notebook-balkendiagramm-erstellen/
- [5]: Findeisen, P. Die Charakterisierung der Normalverteilung nach Gauß. Metrika 29, 55–63 (1982). https://doi.org/10.1007/BF01893364
- [6]: https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/

[ ]: