Schnittstellendokumentation Serie 1 Teil 1

Arsen Hnatiuk,
Max Huneshagen

25. Oktober 2018

Inhaltsverzeichnis

---

1 Einleitung

Aus der Theorie der Taylorentwicklung kann man ein günstiges Verfahren zur Approximation der ersten und zweiten Ableitungen einer Funktion ableiten:

$$f'(x) = \underbrace{\frac{f(x+h) - f(x)}{h}}_{:=D_h^{(1)}(x)} + \mathcal{O}(h) \tag{1}$$

bzw.

$$f''(x) = \underbrace{\frac{f(x+h) - 2f(x) + f(x-h)}{h^2}}_{:=D_h^{(2)}(x)} + \mathcal{O}(h^2) \tag{2}$$

mit der Differenziationsschrittweite h.

Die erstellten Skripte erlauben eine numerische Analyse dieses Verfahrens. Insbesondere wird der absolute Fehler dieses Verfahrens berechnet und sein Verlauf mit der Differenziationsschrittweite h untersucht

2 Schnittstellendokumentation differenzieren.py

Die in *differenzieren.py* enthaltene Klasse `Differenzieren` erlaubt das Plotten von Funktionen und ihren Ableitungen. Die erste und zweite Ableitung kann dabei approximiert und mit der exakten Ableitung verglichen werden.

2.1 Attribute

nicht-statische Attribute:

`ablex_lis` *(list)* :
: Liste aus den ersten drei (anfangend bei 0) exakten Ableitungen der zu untersuchenden Funktion. Der list-Index gibt hierbei den Grad der Ableitung an, wobei die Funktion selbst als nullte Ableitung aufgefasst wird.

`p_arr` *(numpy.ndarray aus floats)* :
: Plotpunkte, an denen die Funktionen geplottet bzw. für die Fehlerbestimmung ausgewertet werden. Das Minimum dieses Arrays wird im Folgenden auch mit a bezeichnet, das Maximum mit b.

2.2 Konstruktor

`Differenzieren(self, fkt, abl_ex, abl2_ex, p_arr)`
Initialisiert ein neues Differenzieren-Objekt. Dazu müssen eine Funktion und die exakte erste und zweite Ableitung übergeben werden.

Input:

`fkt` *(function)* :
: Bestimmt, wohin die Funktionen gezeichnet werden.

`abl_ex` *(function)* :
: Exakte erste Ableitung.

`abl2_ex` *(function)* :
: Exakte zweite Ableitung.

`p_arr` *(numpy.ndarray aus floats)* :
: Plotpunkte, an denen die Funktionen geplottet bzw. für die Fehlerbestimmung aus gewertet werden.

Returns: -

2.3 Methoden

2.3.1 plotfkt_exakt

`plotfkt_exakt(self, plotbereich, grad=0, **kwargs)`
Plottet eine („exakte“) Funktion an den Plotpunkten in einen zu übergebenden Plot. Man kann zwischen nullter, erster und zweiter Ableitung wählen.
Input:
`plotbereich` *(pyplot.Axes-Objekt)* :
Bestimmt, wohin die Funktion geplottet wird.
`grad` *(int, optional, Standard: 0)* :
Grad der Ableitung. Bei grad==0 wird die Funktion selbst geplottet.
`**kwargs` *(keyword arguments, optional)* :
Keyword arguments zur Übergabe an `pyplot.plot`. Nachzulesen in der `matplotlib.lines.Line2D`-Doku.
Returns: -

2.3.2 ablapprox

`ablapprox(self, schrittw, grad=1)`
Diese Funktion approximiert die erste oder zweite Ableitung der Funktion an den plotpunkten. Genutzt wird hierzu die Gleichung (1) bzw. (2). Wird als Grad der zu approximierenden Ableitung 0 angegeben, so wird der Funktionswert an der Plotpunkten zurückgegeben. Zu beachten ist hierbei, das die Funktion auf dem Intervall $[a-h, b+h]$ definiert sein muss. Sollte dies nicht der Fall sein, sollten als Plotpunkte entsprechend modifizierte Arrays gewählt werden. Da aber alle hier sinnvoll zu betrachtenden Funktionen auf ganz $\mathbb{R}$ definiert sein werden, stellt dies im Allgemeinen kaum eine praktische Einschränkung dar. Nennenswert ist außerdem, dass durch die durch (1) gegebene sog. Vorwärtsmethode für sehr große h (i. e. $h \gg {}^{|a-b|}/_{p}$ mit der Anzahl der Plotpunkte p) zu einer „Verschiebung“der approximierten Ableitung und damit zu einer Vergrößerung des Fehlers führt. Dieses Artefakt ist allerdings in der doppelt-logarithmischen Darstellung der Fehler nicht sichtbar. Für kleine h verliert dieser Effekt an Bedeutung, was diese Methode rechtfertigt, da die Konvergenz gegen f' ja ohnehin erst für $h \to 0$ zu erwarten ist.
Input:
`schrittw` *(float)* :
Schrittweite der diskreten Differenziation.
`grad` *(int, optional, Standard: 1)* :
Grad der gewünschten Ableitung. bei `grad==0` werden die Funktionswerte an den Plotpunkten zurückgegeben.
Returns:
(numpy.ndarray aus floats) :
Werte der gewünschten Ableitung der Funktion an den Plotpunkten.

2.3.3 plotfkt_approx

`plotfkt_approx(self, schrittw, plotbereich, grad=1, **kwargs)`
Plottet die approximierte Ableitung (eines bestimmten Grades) der Funktion. Dabei wird die Funktion als nullte Ableitung aufgefasst.
Input:
`schrittw` *(float)* :
Schrittweite der diskreten Differenziation.
`plotbereich` *(pyplot.Axes-Objekt)* :
Bestimmt, wohin die Funktion geplottet wird.
`grad` *(int, optional, Standard: 1)* :
Grad der gewünschten Ableitung. bei `grad==0` wird die Funktion selbst geplottet.
`**kwargs` *(keyword arguments, optional)* :
Keyword arguments zur Übergabe an `pyplot.plot`. Nachzulesen in der `matplotlib.lines.Line2D`-Doku.
Returns: -

2.3.4 err_abl

`err_abl(self, schrittw, grad=1)`
Diese Funktion bestimmt das Maximum der absoluten Differenz zwischen approximierter Ableitung einer Funktion und der exakten Ableitung an den Plotpunkten.
Input:
`schrittw` *(float)* :
Schrittweite der diskreten Differenziation.
`grad` *(int, optional, Standard: 1)* :
Grad der gewünschten Ableitung. bei `grad==0` wird als Fehler 0 zurückgegeben, da die Funktion selbst exakt gegeben ist.
`**kwargs` *(keyword arguments, optional)* :
Keyword arguments zur Übergabe an `pyplot.plot`. Nachzulesen in der `matplotlib.lines.Line2D`-Doku.
Returns:
(float) :
Maximale absolute Abweichung der approximierten Ableitung von `ablex`.

2.4 Weitere Funktionen in differenzieren.py

2.4.1 negsin

`negsin(arg)`
Da python-Funktionen nicht mit Skalaren multipliziert werden können, ist eine separate Definition des negativen Sinus, also der zweiten Ableitung des Sinus erforderlich.
Input:
`arg` *(float)* :
Stelle, die untersucht werden soll.

Returns:

(float) :

Wert von $-\sin$ an der Stelle `arg`.

2.4.2 test

`test()`

Diese Funktion dient dem Testen der Klasse. Sie wird nur ausgeführt, wenn der Nutzer „test“als Kommando beim Ausführen von *differenzieren.py* übergibt. Testweise wird für den Sinus eine Approximation der ersten beiden Ableitungen und der Vergleich mit den exakten Funktionen durchgeführt.

Input: -

Returns: -

3 Bedienungsanleitung zum Hauptprogramm

In der Datei `hauptprogramm.py` wird eine `main()` Methode implementiert. In dieser Methode werden alle Objekte erstellt, die später zur graphischen Darstellung des Fehlerplots dienen. Es werden auch die in der `differenzieren` Klasse definierten Methoden mittels der Beispielfunktion (Sinus) veranschaulicht.

Zuerst wird der Benutzer aufgefordert, eine Schrittweite einzugeben. Es muss eine beliebige nichtnegative `float` Zahl eingegeben werden. Die Gültigkeit des Eingabetyps wird durch `exceptions` behandelt, und danach durch eine `if` Schleife auf Positivität geprüft. Eine nicht gültige Eingabe wird nicht berücksichtigt und der Benutzer wird erneut gefordert, eine Eingabe zu machen.

Diese Kontrolle erfolgt auf die folgende Weise:

```
abbr = 1
while abbr == 1:
    try:
        h_test = float(input('Mit welcher Schrittweite wollen Sie die' +
        ' Ableitungen  approximieren?\n' +
        'Schreiben Sie bitte eine echt positive Zahl, z.B. 0.1\n'))
        abbr = 0
    except ValueError:
        print('Nicht gueltiger Wert eingegeben. Versuchen Sie erneut.')
        abbr = 1
    if h_test <= 0:
        print('Nicht gueltiger Wert eingegeben. Versuchen Sie erneut.')
        abbr = 1
```

Danach wird die Beispielfunktion (der Sinus) bearbeitet. Es werden Die `subplots` und die `arrays` erzeugt, auf welchen die Abbildungen geplottet und entsprechend evaluiert und approximiert werden. Ein Objekt der `differenzieren` Klasse für die Sinus Abbildung wird ebenfalls erzeugt. Schließlich wird die Sinus Abbildung samt seiner ersten beiden Ableitungen und der Approximation (für die durch den Benutzer eingegebene Schrittweite) dieser Ableitungen mittels der in der `Differenzieren`-Klasse definierten

Methoden graphisch dargestellt. Der absolute Fehler der Approximation jeder wird jeweils in der Standardausgabe angezeigt.

Anschließend wird die Funktion `fehlerplot()` aufgerufen, um die Beziehung zwischen dem absoluten Fehler und der Schrittweite zu veranschaulichen.

3.1 fehlerplot()

In dieser Funktion wird ein Plot des absoluten Fehlers in Abhängigkeit von der Schrittweite mittels der Differenzieren Klasse gezeichnet. Diese Funktion benutzt insbesondere die `numpy.vectorize`-Methode, um Arrays mit den Fehlerwerten zu erzeugen, und die `numpy.loglog`-Methode, um eine doppelt-logarithmisch skalierte Graphik zu erstellen.

Input:

`plotbereich` *(pyplot.Axes-Objekt)* Subplot, auf dem der Plot erzeugt wird

`diff_objct` *(Differenzieren-Instanz)* Differenzieren-Objekt, das untersucht wird

`h_array` *(numpy.ndarray aus floats)* Array mit den Schrittweiten

Returns: -

3.2 Beispieldurchlauf

Im Folgenden wird ein Beispieldurchlauf von `main()` demonstriert. Der eingegebene Wert für die Schrittweite ist 0.1.

```
Mit welcher Schrittweite wollen Sie die Ableitungen approximieren?
Schreiben Sie bitte eine echt positive Zahl, z.B. 0.1
0.1
Der absolute Fehler in der ersten Ableitung ist 0.04999, Schrittweite 0.1
Der absolute Fehler in der zweiten Ableitung ist 0.00083, Schrittweite 0.1
```

Es werden dazu zwei Graphiken erzeugt (Abbildung 1). Man sieht auf der linken Seite eine Graphik mit dem absoluten Fehler als Funktion von der Schrittweite. Auf der rechten Graphik steht die Beispielfunktion mit ihren Ableitungen (exakt und approximiert). Man erkennt für diesen vergleichsweise großen h-Wert noch leicht die oben erwähnte „Verschiebung“der approximierten ersten Ableitung gegenüber der exakten Funktion. Der damit verbundene Fehler ist aber bereits hier hinnehmbar klein und lässt sich optisch schon ab $h = 0.01$ nicht mehr wahrnehmen, während h hier ohne Weiteres bis zu $h = 10^{-10}$ verkleinert werden kann (s. Abbildung 1).

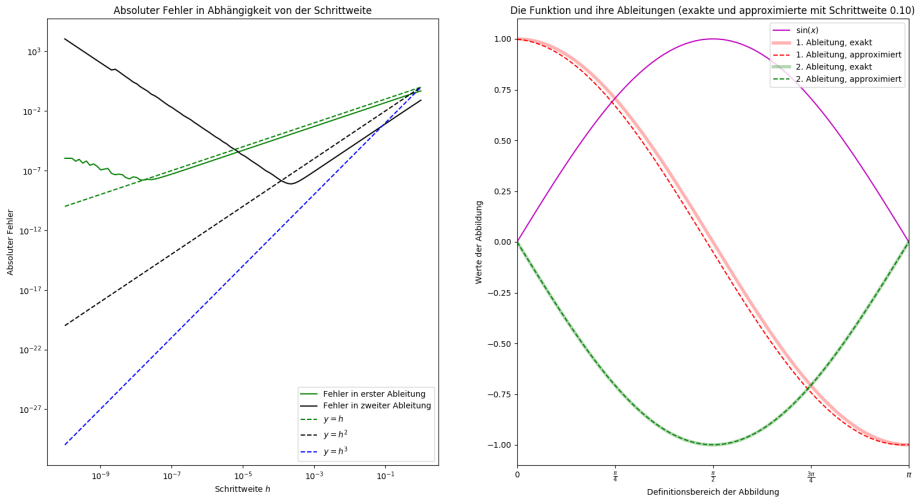

Abbildung 1: Die ausgegebene Graphik für $h = 0.1$