Semesterarbeit Teil 3a: Newton-Verfahren

Aufgabenstellung

Implementieren Sie das Newton-Verfahren zur Bestimmung von Nullstellen von Funktionen.

(Dabei soll die Funktion als Sympy-Objekt eingegeben werden, so dass Sie mit Sympy die Ableitung davon bestimmen können.)

Implementation

Struktur

Für diese Arbeit verwende ich folgende Scripts:

- «test.py»: Dieses Script dient zur Überprüfung der Implementationen.
- «tools.py»: Dieses Script enthält die Funktion mit dem Newton-Verfahren.

Test-Script

Im Test-Script habe ich die Aufgabe 1 aus [Mk] Seite 291 Newtonverfahren und Taylorentwicklung gelöst:

```
Die Nullstelle fuer `-x + 3*cos(x)=0` bei `x0=1` liegt bei `1.17012095000756`.
```

Dafür importiere ich die `sympy`-Library und das Tools-Script mit der Newton-Verfahren-Funktion:

```
import sympy
import tools
```

Dann erstelle ich ein sympy-Symbol `x`, welches dann später in den sympy-Objekten den X-Wert darstellt:

```
x = sympy.Symbol('x')
```

Nun kann ich das sympy-Objekt erstellen für `3cos(x) – x = 0`:

```
demoFunction = 3 * sympy.cos(x) - x
demoDerivation = sympy.diff(demoFunction)
```

Mit Hilfe der Funktion `sympy.diff` kann ich dann die abgeleitete Funktion berechnen. Diese brauche ich für das Newton-Verfahren.

```
demoX0 = 1
```

Ich speichere noch den x0-Wert in eine Variable und führe dann die Funktion mit dem Newton-Verfahren auf:

```
demoZero = tools.newtons_method(demoX0, demoFunction, demoDerivation, 5)
```

Zum Schluss gebe ich noch die Nullstelle, also `demoZero` zurück:

```
print('Die Nullstelle fuer `' + str(demoFunction) + '=0` bei `x0=' + str(demoX0)
      + '` liegt bei `' + str(demoZero) + '`.')
```

Tools-Script

In diesem Script geschehen die mathematisch interessanten Teile des Skriptes. Als erstes importiere ich die sympy-Library und erstelle wieder den symbolischen X-Wert, damit ich diesen für meine Berechnungen verwenden kann:

```
import sympy

x = sympy.Symbol('x')
```

Dann definiere ich die Funktion mit 5 Parameter:

```
def newtons_method(x0, function, derivation, decimals, calls=1000):
```

- `x0`: definiert den x0-Wert
- `function`: definiert die Funktion
- `derivation`: definiert die Ableitung der Funktion
- `decimals`: definiert die Anzahl Dezimalstellen, welche übereinstimmen müssen, also die Genauigkeit
- `calls`: definiert die maximale Anzahl an Aufrufen, welches das Script noch durchlaufen darf, da das Script sich immer wieder selbst aufruft und es möglich wäre, dass keine Nullstelle existiert. In diesem Fall würde die Funktion `None` zurückgeben. Der Standardwert für `calls` liegt bei 1000 Durchführungen.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Gemäss Newton-Verfahren gilt: , deshalb berechnen wir x1 basierend auf x0:

```
    x1 = x0 - (function.evalf(subs={x: x0}) / derivation.evalf(subs={x: x0}))
```

Da wir sympy-Objekte haben, können wir dafür ganz einfach die `evalf`-Funktion verwenden und den X-Wert einsetzen.

Als nächstes vergleichen wir x0 mit x1. Je näher dieser Delta-Wert an 0 kommt, desto näher sind wir unserer Nullstelle:

```
    diff_delta_x = abs(x0 - x1)
    diff_max_decimals = 1 / (10 ** decimals) # => 0.00001
```

Normalerweise wird jedoch dieser Wert nicht nie absolut 0 sein, deshalb haben wir die `decimals`-Variable. Deshalb überprüfen wir, wie viele Dezimalstellen bei x0 und x1 identisch sind (anhand des Delta-Wertes). Wenn die Anzahl identischer Dezimalstellen mindestens dem Wert des Parameters `decimals` entspricht geben wir x1 zurück:

```
    if diff_delta_x < diff_max_decimals: # 0.00000999… < 0.00001 => also 5 identische Stellen
        return x1
```

Ansonsten rufen wir die Funktion rekursiv, erneut auf (falls die `calls`-Begrenzung noch nicht ausgelaufen ist):

```
    if calls > 0:
        return newtons_method(x1, function, derivation, decimals, calls - 1)
    else:
        return None
```