Praktikum Wissenschaftliches Rechnen

TECHNISCHE UNIVERSITÄT DARMSTADT

Interpolation

Sebastian Schöps, Timon Seibel

Sommersemester 2025
28. April 2025
Übungsblatt 1

Ziel dieser Übung ist es, die Newton-Interpolation und die Interpolation mit kubischen Splines in `Python` zu implementieren. Hierzu wird Ihnen auf `Moodle` das Grundgerüst des abzugebenden `Python`-Projekts zur Verfügung gestellt. Ihre Aufgabe ist es, den Code entsprechend der Aufgabenstellung fertigzustellen.
Sie dürfen weitere (Hilfs-)Methoden hinzufügen, um Ihren Code übersichtlicher zu gestalten. Aus der Sicht des Software Engineerings wird dies sogar empfohlen. Kommentieren Sie Ihren Code, sodass er für Dritte verständlich ist.

Wichtig: Die vorgegebenen (noch zu vervollständigenden) Methoden sind zwingend zu verwenden. Insbesondere dürfen weder die Namen der vorgegebenen Methoden noch die Reihenfolge ihrer Argumente abgeändert werden. Ebenso dürfen Sie die Rückgabewerte nicht ändern. Andernfalls könnten die zugehörigen Tests fehlschlagen, was zu Punktverlust führt.
Weiterhin möchten wir Sie daran erinnern, dass Sie, mit Ausnahme von `numpy` und `matplotlib`, keine externen Bibliotheken benutzen dürfen, sofern dies die Aufgabenstellung nicht ausdrücklich erlaubt.

Zur Bearbeitung der Übung stellen wir Ihnen die folgenden Dateien zur Verfügung:

interpolation.py Grundgerüst der Implementierung der Newton- und Spline-Interpolation

main.py Von hieraus sollen die geforderten Plots erstellt und gespeichert werden

Abgabe: Komprimieren Sie alle relevanten und Verzeichnisse in `gruppe_xy_uebung_1.zip` (oder `.tar.gz`), wobei `xy` durch Ihre Gruppennummer zu ersetzen ist, und reichen Sie die Archiv-Datei bis **spätestens 11.05.2025, 23:59 Uhr** auf `Moodle` ein. Spätere Abgaben können nicht berücksichtigt werden. Reichen Sie Ihre Abgabe daher frühzeitig ein und melden Sie sich bei Problemen mit dem Upload vor Ablauf der Deadline, damit wir gegebenenfalls eine Lösung finden können.

Aufgabe 1.1: Newton-Interpolation (6 Punkte)

In dieser Aufgabe sollen Sie die Newton-Interpolation (siehe Abschnitt 1.1.2 im Skript) mit äquidistanten sowie Tschebyschow-Stützstellen (siehe (1.7) im Skript) implementieren. Hierfür sollen Sie die Methoden `newton_equidistant` und `newton_chebyshev` vervollständigen. Beide Methoden erwarten als Input den Parameter `func` vom Typ `Callable[[float], float]`, dies ist eine Funktion, welche mit dem Aufruf `func(x)` eine Zahl als Ergebnis liefert, wobei `x` eine Zahl ist[1]. Weiterhin verlangen die Methoden die Übergabe der Intervallgrenzen `bnds` in der Form `[a,b]`, die Anzahl an Auswertungsstellen `n_eval_pts`, sowie die Anzahl an Stützstellen `n_sample_pts`. Als Rückgabewerte liefern beide Methoden die Tupel von `numpy.ndarrays [x,y]` und `[xx,yy]`. Hierbei bezeichnen `x,y` die Arrays mit den Auswertungspunkten und den Werten des Interpolationspolynoms an diesen Stellen. Mit `xx,yy` bezeichnen wir die Stützstellen und -werte.
Unabhängig von der Methode sollen Sie für die Auswertungsstellen `x` eine äquidistante Verteilung mit `n_eval_pts` Stellen wählen, wobei die Intervallgrenzen selbst auch Auswertungsstellen sind.
Für die Stützstellen `xx` sollen Sie entsprechend dem Methodennamen `n_sample_pts` äquidistante oder Tschebyschow Stützstellen wählen.

[1] Bei `x` kann es sich auch um ein `numpy.ndarray` handeln, wobei `func(x)` dann wiederum ein `numpy.ndarray` zurückgibt.

Aufgabe 1.2: Spline-Interpolation (4 Punkte)

Stellen Sie als nächstes die Methoden `cubic_spline_equidistant` und `cubic_spline_chebyshev` fertig. Diese interpolieren auf äquidistanten bzw. Tschebyschow Stützstellen mit kubischen Splines. Input- und Outputvariablen sind die gleichen wie in Aufgabe 1.1. Jedoch sollen Sie die Interpolation diesmal nicht selbst implementieren, sondern auf eine Bibliothek zurückgreifen und beispielsweise `scipy.interpolate.CubicSpline` nutzen.

Aufgabe 1.3: Plots (10 Punkte)

Abschließend sollen Sie in der Datei `main.py` die Funktion $f(x) = \frac{1}{1+x^2}$ mit jeder Ihrer vier Methoden auf dem Intervall $[-5, 5]$ interpolieren lassen und die Ergebnisse graphisch[2] darstellen. Verwenden Sie für die Interpolation jeweils 7 Stützstellen und 1000 Auswertungspunkte. Erstellen Sie für jede der vier Interpolationen einen separaten Plot, in welchem Sie die analytische Funktion und die interpolierte Funktion plotten lassen. Markieren Sie zusätzlich die Stützpunkte mit einem `X`.
Beschriften Sie die Achsen korrekt, nennen Sie die genaue Interpolation im Titel und fügen Sie eine Legende hinzu, welche die analytische Funktion, die interpolierte Funktion sowie die Stützpunkte markiert. Lassen Sie die Plots von `matplotlib` unter den Namen `Interpolation_Newton_equi.pdf`, `Interpolation_Newton_cheb.pdf`, `Interpolation_Cubic_Spline_equi.pdf` bzw. `Interpolation_Cubic_Spline_cheb.pdf` in einen Ordner names `Plots` exportieren[3].

Als letzten Schritt sollen Sie einen y-Semilog-Plot[4] der relativen Fehler erstellen lassen. Berechnen sie hierfür den relativen Fehler $\frac{|\texttt{func(x)}-\texttt{y}|}{|\texttt{func(x)}|}$ jeder Interpolation, wobei `y` die zurückgegebenen Werte der Interpolation sind und `func(x)` die Funktion `func` ausgewertet an jedem Auswertungspunkt in `x` ist. Stellen Sie die Fehler der vier Interpolationen alle in dem selben Plot dar. Wählen Sie für diesen Plot eine lineare x-Achse und eine logarithmische y-Achse[5]. Ergänzen Sie den Plot um Achsenbeschriftungen und eine Legende. Lassen Sie den Plot von `matplotlib` unter dem Namen `Interpolation_Rel_Errors.pdf` exportieren.

Aufgabe 1.4: Tests

Damit Sie selbst einen Eindruck davon erhalten können wie automatisierte Tests in `Python` aussehen, stellen wir Ihnen die Datei `test.py` zur Verfügung, welche bereits Beispieltests beinhaltet. Diese überprüfen lediglich, dass Ihre Implementierung den korrekten Input entgegennimmt und der Output das richtige Format hat. Gerne dürfen Sie für sich selbst weitere Tests ergänzen, um Ihre Implementierung hiermit zu überprüfen. Das wird aber nicht bewertet.

[2] Wir empfehlen die Verwendung von `matplotlib.pyplot` (`https://matplotlib.org/`)
[3] In `matplotlib.pyplot` können Sie dies mittels `savefig('Plots/PLOTNAMEpdf', dpi=300)` erreichen.
[4] z.B. `matplotlib.pyplot.semilogy`
[5] Um einen aussagekräftigeren Plot zu erhalten, können Sie als untere Grenze für die y-Achse 10^{-6} wählen.