Praktikum Wissenschaftliches Rechnen

Gewöhnliche Differentialgleichungen

Sebastian Schöps, Timon Seibel

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2025
26. Mai 2025
Übungsblatt 3

Ziel dieser Übung ist es, verschiedene Differenzenquotienten sowie das explizite und implizite Euler-Verfahren 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 Übung ist **in Gruppen** zu bearbeiten und 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:

ode.py Implementierung der Differenzenquotienten und des Euler-Verfahrens

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

Abgabe: Komprimieren Sie alle relevanten Verzeichnisse in `gruppe_xy_uebung_3.zip` (oder `.tar.gz`), wobei `xy` durch Ihre Gruppennummer zu ersetzen ist, und reichen Sie die Archiv-Datei bis **spätestens 08.06.2025, 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 3.1: Differenzen-Quotienten (6 Punkte)

Die Ableitung $y'(x)$ einer Funktion $y(x)$ kann numerisch durch Differenzenquotienten approximiert werden. Zu den gängisten zählen der vorwärtsgerichtete, rückwärtsgerichtete und zentrale Differenzenquotient. Für gegebene Schrittweite h approximieren diese y' an der Stelle x wie folgt:

$$\begin{aligned}
\text{Vorwärts:} \quad & y'(x) \approx \frac{y(x+h) - y(x)}{h} \\
\text{Rückwärts:} \quad & y'(x) \approx \frac{y(x) - y(x-h)}{h} \\
\text{Zentral:} \quad & y'(x) \approx \frac{y(x+h) - y(x-h)}{2h}
\end{aligned}$$

Ihre Aufgabe ist es nun, diese drei Varianten des Differenzenquotienten zu implementieren. Vervollständigen Sie hierzu die Methoden `forward_difference_quotient`, `backward_difference_quotient` und `central_difference_quotient`. Als Input erwartet jede der drei Methoden eine Funktion `f` vom Typ `Callable[[float], float]`, eine Stelle `x: float` sowie die Schrittweite `h: float`. Als Output gibt jede Methode

die jeweilige Approximation von `f'` and der Stelle `x` als `float` zurück.
Neben der Berechnung des jeweiligen Differenzenquotienten soll jede Methode zunächst überprüfen, dass die Schrittweite `h` positiv ist und im Falle $h \leq 0$ einen `ValueError`[1][2] werfen.

Aufgabe 3.2: Euler-Verfahren (4 Punkte)

Als nächstes sollen Sie das implizite und explizite Euler-Verfahren implementieren (siehe 3.1.2 im Skript). Vervollständigen Sie hierzu die Methoden `explicit_euler` und `implicit_euler`. Beide Methoden erwarten als Input die rechte Seite `rhs: Callable[[float],float]` der ODE $y'(t) = rhs(t, y)$, den Startwert `y0: float` entsprechend der Randbedingung $y_0 = y(t_0)$, den Startzeitpunkt `t0: float`, den Endzeitpunkt `T: float` und die Anzahl an Zeitschritten `N: int`. Als Rückgabewerte erhalten Sie jeweils das Array `t: numpy.ndarray[float]` der Zeitpunkte $(t_i)_i$ sowie das Array `y: numpy.ndarray[float]` der Approximation $y_i \approx y(t_i)$.

Hinweis: In jedem Schritt des impliziten Euler-Verfahrens muss die Gleichung $y_{i+1} = y_i + h \cdot rhs(t_i, y_{i+1})$ nach y_{i+1} gelöst werden. Für diesen Schritt sollen Sie einen Löser wie `scipy.optimize.fsolve` nutzen.

Aufgabe 3.3: Plots (10 Punkte)

Abschließend sollen Ihre soeben implementierten Methoden verwenden, um verschiedene Plots erzeugen zulassen. Denke Sie wieder daran, Ihre Achsen korrekt zu beschrift, einen Titel sowie eine Legende hinzuzufügen und abschließend den Plot in den Ordner `Plots` exportieren zu lassen.

3.3a)

Betrachten Sie zunächst die Funktion $f(x) = \sin(x)$ auf dem Intervall $[0, 2\pi]$. Berechnen Sie auf diesem Intervall in 100 äquidistanten Punkten mit jeder der drei Methoden jeweils die Approximation von f' mit einer Schrittweite von $h = 0.2$. Plotten Sie die mit dem vorwärts, rückwärts und zentralen Differenzenquotienten approximierten Ableitungen als Punkte (nicht als durchgezogene Linien) im selben Plot. Bestimmen Sie als Referenz die analytische Ableitung f' und fügen Sie diese dem Plot als durchgezogene Linie hinzu (achten Sie darauf, dass die gepunkteten Linien weiterhin sichtbar sind). Lassen Sie den Plot als `Difference_Quotient_Comparison.pdf` exportieren.

3.3b)

Betrachten Sie als nächstes die gewöhnliche Differentialgleichung (ODE)

$$\begin{aligned} y'(t) &= \frac{1}{2}\left(1 - \frac{y(t)}{8}\right) \cdot y(t) =: f(t, y), \\ y(0) &= 0.25. \end{aligned}$$

Durch Einsetzen lässt sich leicht zeigen, dass diese ODE durch

$$y(t) = \frac{2}{0.25 + 7.75\exp(-0.5t)}$$

gelöst wird.
Berechnen Sie, jeweils mit dem expliziten als auch impliziten Euler-Verfahren, die approximative Lösung für $n = 100,\ 500,\ 1\,000,\ 5\,000,\ 10\,000$ Zeitschritte auf $[0, 10]$. Berechnen Sie den Fehler $|y(t_n) - y_n|$ in $t = 10$ für jedes n und plotten Sie den Fehler in doppellogarithmischer Darstellung über n. Wählen Sie für das explizite Euler-Verfahren eine durchgezogene Linie und für das implizite eine gestrichelte. Exportieren Sie den Plot als `Euler_Error_VS_N.pdf`.

[1] `https://docs.python.org/3/library/exceptions.html`
[2] `https://www.digitalocean.com/community/tutorials/python-valueerror-exception-handling-examples`

Berechnen Sie weiterhin, sowohl für das explizite als auch das implizite Euler-Verfahren, für $n = 10\,000$ Zeitschritte den Fehler $|y(t_i) - y_i|$ in Abhängigkeit von der Zeit und plotten Sie den Fehler über der Zeit t. Wählen Sie für das explizite Euler-Verfahren eine durchgezogene Linie und für das implizite eine gestrichelte. Exportieren Sie den Plot als `Euler_Error_VS_Time.pdf`.

3.3c)

Betrachten Sie abschließend die ODE

$$y'(t) = -4(y(t) - 2) =: f(t, y),$$
$$y(0) = 1,$$

welche durch $y(t) = 2 - \exp(-4t)$ gelöst wird. Lösen Sie die ODE jeweils mit dem expliziten als auch dem impliziten Euler-Verfahren auf dem Zeitintervall $[0, 5]$. Verwenden Sie hierbei je einmal eine Schrittweite von $h = 0.2$ (dies entspricht $n = 25$) und $h = 0.5$ ($n = 10$). Erstellen Sie nun einen Plot mit zwei Subplots[3] nebeneinander, wobei sie im linken Subplot die Ergebnisse des expliziten Euler-Verfahrens und im rechten Subplot die Ergebnisse des impliziten Euler-Verfahrens darstellen soll. Plotten Sie je Subplot die analytische Lösung sowie jeweils die zugehörige approximierte Lösung für $h = 0.2$ und $h = 0.5$. Exportieren Sie den Gesamtplot als `Comparison_Explicit_Implicit_Euler.pdf`.

Hinweis: Ziel dieser Aufgabe ist es, die Instabilität des expliziten Euler-Verfahrens für zu große Schrittweiten zu visualisieren. Dies bedeutet, dass die mit dem expliziten Euler-Verfahren für $h = 0.5$ approximierte Lösung um die analytische Lösung oszilliert statt zu konvergieren. Die drei anderen Approximationen hingegen konvergieren. Sollten Sie keine Oszillation feststellen, erhöhen Sie die Schrittweite (wählen Sie also $n < 10$) und überprüfen Sie, ob die approximierte Lösung zu oszillieren beginnt. Ist dies nicht der Fall, so liegt wahrscheinlich ein Fehler in Ihrer Implementierung vor.

Aufgabe 3.4: 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.

[3] Dies können sie mittels `matplotlib.pylot.subplot(1,2,i)` erreichen, wobei Sie mit `i=1` den linken und mit `i=2` den rechten Subplot auswählen. Rufen Sie einfach `subplot` auf und fahren Sie dann wie gewohnt mit dem Plotten fort.