Praktikum Wissenschaftliches Rechnen

Quadratur

Sebastian Schöps, Timon Seibel

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2025
12. Mai 2025
Übungsblatt 2

Ziel dieser Übung ist es, die Newton-Cotes- und die Monte-Carlo-Quadratur 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:

quadrature.py Grundgerüst der Implementierung der Newton-Cotes- und Monte-Carlo-Quadratur

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

beobachtungen.txt Hier sollen sie Ihre Beobachtungen schriftlich festhalten

Abgabe: Komprimieren Sie alle relevanten und Verzeichnisse in `gruppe_xy_uebung_2.zip` (oder `.tar.gz`), wobei `xy` durch Ihre Gruppennummer zu ersetzen ist, und reichen Sie die Archiv-Datei bis **spätestens 25.05.2025, 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 2.1: Newton-Cotes-Quadratur (6 Punkte)

In dieser Aufgabe sollen Sie die geschlossene Newton-Cotes-Quadratur mit variabler Stützstellenzahl implementieren. Es soll sowohl die lokale als auch summierte Quadratur implementiert werden (siehe Kapitel 2 im Skript).

2.1a)

Essentiell für die Berechnung der Newton-Cotes-Quadratur sind die zugehörigen Gewichte. Vervollständigen Sie daher die Methode `newton_cotes_weights`, welche den Polynomgrad `n: int` des Interpolationspolynoms übergeben bekommt und das Array `weights: numpy.ndarray[float]` der entsprechenden Quadraturgewichte zurückgibt. Für die hierbei benötigte Integration der Lagrangepolynome dürfen Sie z.B. `scipy.integrate.quad`[1] verwenden.

[1] Verwenden Sie die Default-Werte, sprich, übergeben Sie lediglich die zu integrierende Funktion `func` sowie Start- und Endwert `a` und `b`, jedoch keines der optionalen Argumente.

2.1b)

Als nächstes sollen Sie die Methode `newton_cotes_quadrature` für die **lokale** Newton-Cotes-Quadratur vervollständigen. Die Methode erwarten als Input die Funktion `f` als Parameter vom Typ `Callable[[float], float]`[2], die Integrationsgrenzen `a: float` und `b: float` sowie den Grad `n: int` der Quadratur. Als Rückgabewerte liefert die Methode den approximierten Wert des Integrals als `float`.

2.1c)

Vervollständigen Sie im Anschluss die Methode `global_newton_cotes_quadrature` für die **summierte** Newton-Cotes-Quadratur. Neben den Eingabewerten von `newton_cotes_quadrature` erwartet die Methode zusätzlich noch die Anzahl an Teilintervallen `m: int`. Als Rückgabewerte liefert die Methode den approximierten Wert des Integrals als `float`.

Aufgabe 2.2: Monte-Carlo-Quadratur (4 Punkte)

Als weitere Quadratur sollen Sie sich mit der Monte-Carlo-Quadratur befassen. Die Monte-Carlo-Quadratur verwendet die Approximation

$$\int_a^b f(x)\mathrm{d}x \approx \tfrac{(b-a)}{n} \sum_{i=1}^{n} f(x_i),$$

wobei die Stützstellen $(x_i)_{1\leq i\leq n}$ n gleichverteilte Zufallszahlen im Intervall $[a, b]$ sind (verwenden Sie z.B. `numpy.random.uniform` zur Bestimmung der Stützstellen). Da es sich um ein probabilistische Verfahren handelt, benötigt die Monte-Carlo-Quadratur eine sehr große Zahl an Stützstellen, um genaue Resultate zu liefern.
Vervollständigen Sie die Methode `monte_carlo_quadrature`. Diese erwartet als Eingabewert die zu integrierende Funktion `f` als Parameter vom Typ `Callable[[float],float]`, die Integrationsgrenzen `a: float` und `b: float` sowie die Stichprobengröße `n: int`. Als Rückgabewerte liefert die Methode den approximierten Wert des Integrals als `float`.

Aufgabe 2.3: Plots (10 Punkte)

Abschließend sollen Sie in der Datei `main.py` das Integral der Funktion $f(x) = \cos(x) + e^x + 3x^2$ auf $[0, 5]$ mit der Newton-Cotes- und Monte-Carlo-Quadratur approximieren lassen und die Ergebnisse graphisch darstellen. Denken Sie daran, die Achsen korrekt zu beschriften und fügen Sie eine Legende sowie einen Titel hinzu. Speichern Sie jeden Plot in einer separaten PDF-Datei unter dem vorgegebenen Dateinamen im Ordner `Plots`.
Verwenden Sie die Datei `beobachtungen.txt`, um Ihre Beobachtungen schriftlich festzuhalten. Ein bis zwei Sätze pro Experiment sind hierbei vollkommen ausreichend.

2.3a)

Zunächst sollen Sie sich das Verhalten der lokalen Newton-Cotes-Quadratur für verschiedene Ordnungen anschauen. Berechnen Sie hierfür $I_n(x) \approx \int_0^5 f(x)\mathrm{d}x$ für die Ordnungen $n = 1, \ldots, 20$ und plotten Sie $I_n(x)$ über n. Fügen Sie dem Plot eine gestrichelte, graue Linie hinzu, welche parallel zur `x`-Achse auf Höhe des analytischen Integralwerts verläuft. Lassen Sie den Plot als `Newton_Cotes.pdf` exportieren und halten Sie Ihre Beobachtungen in `beobachtungen.txt` schriftlich fest.
Wiederholen Sie das Experiment für die Monte-Carlo-Quadratur. Da die Monte-Carlo-Quadratur jedoch prinzipiell viele Stützstellen benötigt, betrachten Sie diesmal $n = 1, \ldots, 10\,000$ Stützstellen. Lassen Sie den Plot als `Monte_Carlo.pdf` exportieren und halten Sie Ihre Beobachtungen in `beobachtungen.txt` schriftlich fest.

[2] Denken Sie daran, dass Sie an `f` auch ein `numpy.ndarray[float]` übergeben können und ein `numpy.ndarray[float]` zurückbekommen.

Hinweis: Da es sich bei der Monte-Carlo-Quadratur um ein probabilistisches Verfahren handelt, erhalten Sie bei mehrfacher Ausführung stets unterschiedliche Ergebnisse. Sie können jedoch vor Aufruf der Monte-Carlo-Quadratur mit `numpy.random.seed` einen Seed festsetzen, sodass Sie stets dieselben Ergebnisse erhalten. Nötig ist das für die Plots aber nicht, denn wir betrachten nur das allgemeine Verhalten, welches seedunabhängig ist.

2.3b)

Als nächstes sollen Sie für die drei implementierten Verfahren Konvergenzplots erstellen. Verwenden Sie für jeden Konvergenzplot eine doppellogarithmische Darstellung[3] und zeigen Sie auch jeweils, im selben Plot, die Kurve der theoretisch zu erwartenden Konvergenz als Vergleich. Stellen Sie hierfür die berechneten Fehler mit Hilfe von Markern (Kreuze, Kreise, Rauten, etc.), jedoch ohne Linien, dar. Die theoretische Konvergenzkurve soll so geplottet werden, dass die Marker möglichst auf der Kurve liegen. Nutzen Sie f aus der vorherigen Aufgabe für die Fehlerberechnung.

1. Berechnen Sie den Fehler der lokalen Newton-Cotes-Quadratur auf den Intervallen $[0, h_j]$ mit $h_j = \frac{5}{2^j}$, $j = 0, \dots, 9$ jeweils für die Trapez- und die Simpsonregel und plotten Sie den Fehler über der Intervalllänge. Stellen Sie die Fehler beider Regeln im selben Plot dar und lassen Sie ihn als `Local_Convergence_Plot.pdf` exportieren. Halten Sie Ihre Beobachtungen schriftlich in `beobachtungen.txt` fest.
2. Berechnen Sie den Fehler der summierten Newton-Cotes-Quadratur auf $[0, 5]$ für die Intervalllängen $h_m = \frac{5}{m}$, $m = 1, \dots, 100$ jeweils für die Trapez- und die Simpsonregel und plotten Sie den Fehler über der Intervalllänge. Stellen Sie die Fehler beider Regeln im selben Plot dar und lassen Sie ihn als `Global_Convergence_Plot.pdf` exportieren. Halten Sie Ihre Beobachtungen schriftlich in `beobachtungen.txt` fest.
3. Berechnen Sie den Fehler der Monte-Carlo-Quadratur auf $[0, 5]$ für $n = 1, \dots, 10\,000$ Stützstellen und plotten Sie den Fehler über der Anzahl der Stützstellen. Für die Monte-Carlo-Methode ist eine Konvergenz von $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ zu erwarten. Lassen Sie den Fehler als `Monte_Carlo_Convergence_Plot.pdf` exportieren. Halten Sie Ihre Beobachtungen schriftlich in `beobachtungen.txt` fest.

Aufgabe 2.4: 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] z.B. `matplotlib.pyplot.loglog`