Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Wintersemester 2009/10
13. Jänner 2010

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Die Aufgaben mit Stern (*) sind bis zur nächsten Übung vorzubereiten und werden dort abgeprüft. Kopieren Sie bitte den Source-Code in ein Unterverzeichnis `serie10` *Ihres Home-Verzeichnisses. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit* `matlab` *auf der* `lva.student.tuwien.ac.at` *interpretiert werden können.*

Aufgabe 10.1*. Für einen stetigen Integranden $f : [a, b] \to \mathbb{R}$ berechnet man das Integral $I := \int_a^b f\,dx$ numerisch über geeignete Summen. Bei der *summierten Rechteckregel* berechnet man für gegebenes $n \in \mathbb{N}$ und $h := (b-a)/n$ zum Beispiel

$$I_n := h \sum_{j=1}^{n} f(a + jh). \tag{1}$$

Man schreibe eine Funktion `rechteckregel(f,a,b,tau)`, die die Folge der Approximationen I_n berechnet, bis gilt

$$|I_n - I_{n-1}| \le \begin{cases} \tau & \text{für } |I_n| \le \tau, \\ \tau\,|I_n| & \text{anderenfalls.} \end{cases}$$

In diesem Fall gebe man die vollständige Folge $(I_1, \dots, I_n)$ der Approximationen zurück. Man teste die numerische Integration am Beispiel $f(x) = \exp(x)$ auf dem Intervall $[0, 10]$ und gebe abhängig von n den Fehler $|I - I_n|$ tabellarisch aus.

Aufgabe 10.2*. Zu gegebenen reellen Stützstellen $x_1 < \cdots < x_n$ und Funktionswerten $y_j \in \mathbb{R}$ garantiert die Lineare Algebra ein eindeutiges Polynom $p(t) = \sum_{j=1}^{n} a_j t^{j-1}$ vom Grad $n-1$ mit $p(x_j) = y_j$ für alle $j = 1, \dots, n$. Nun sei $t \in \mathbb{R}$ fixiert und $p(t)$ gesucht. Man kann $p(t)$ mit dem *Neville-Verfahren* berechnen, ohne zunächst den Koeffizientenvektor $a \in \mathbb{R}^n$ berechnen zu müssen: Dazu definiere man für $j, m \in \mathbb{N}$ mit $m \ge 2$ und $j + m \le n + 1$ die Werte

$$\begin{aligned} p_{j,1} &:= y_j, \\ p_{j,m} &:= \frac{(t - x_j)p_{j+1,m-1} - (t - x_{j+m-1})p_{j,m-1}}{x_{j+m-1} - x_j}. \end{aligned}$$

Es gilt dann $p(t) = p_{1,n}$. Man schreibe eine Funktion `neville`, die den Auswertungspunkt $t \in \mathbb{R}$ sowie die Vektoren $x, y \in \mathbb{R}^n$ übernimmt und $p(t)$ mittels Neville-Verfahren berechnet. Dazu berücksichtige man das folgende schematische Vorgehen

$$\begin{array}{ccccccccccccc} y_1 & = & p_{1,1} & \longrightarrow & p_{1,2} & \longrightarrow & p_{1,3} & \longrightarrow & \cdots & \longrightarrow & p_{1,n} & = & p(t) \\ & & & \nearrow & & \nearrow & & & & \nearrow & & & \\ y_2 & = & p_{2,1} & \longrightarrow & p_{2,2} & & & \nearrow & & & & & \\ & & & \nearrow & & & & & & & & & \\ y_3 & = & p_{3,1} & \longrightarrow & \vdots & & & & & & & & \\ \vdots & & \vdots & & \vdots & \nearrow & & & & & & & \\ y_{n-1} & = & p_{n-1,1} & \longrightarrow & p_{n-1,2} & & & & & & & & \\ & & & \nearrow & & & & & & & & & \\ y_n & = & p_{n,1} & & & & & & & & & & \end{array} \tag{2}$$

Der mathematische Beweis für diesen Algorithmus folgt in der Vorlesung zur Numerischen Mathematik. Zunächst schreibe man die Funktion so, dass die Matrix $(p_{j,m})_{j,m=1}^{n}$ vollständig aufgebaut wird. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 10.3*. Eine effiziente Implementierung des einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 9.6 verwendet die vorherigen Werte $\Phi(h_0), \dots, \Phi(h_n)$, indem man (theoretisch!) das Interpolationspolynom p_n vom Grad $n-1$ zu den Punkten $(h_j, \Phi(h_j))$ für $j = 1, \dots, n$ betrachtet, d.h. $p_n(h) \approx \Phi(h)$, und dieses mit dem Neville-Verfahren bei $h = 0$ auswertet. Man bezeichnet dieses Vorgehen als *Richardson-Extrapolation des einseitigen Differenzenquotienten.* (Einen Konvergenzbeweis für dieses Verfahren sehen Sie in der Vorlesung zur Numerischen Mathematik.) Mit $h_n := 2^{-n} h_0$ betrachten wir die Folge der $y_n := p_n(0)$. Man schreibe eine Funktion `richardson`, die neben dem Funktionshandle einer Funktion f, den Auswertungspunkt x, die erste Schrittweite $h_0 > 0$ sowie die Toleranz $\tau > 0$ übernimmt und $y_{n+1} \approx f'(x)$ zurückliefert, sobald gilt

$$|y_n - y_{n+1}| \le \begin{cases} \tau, & \text{falls } |y_{n+1}| \le \tau, \\ \tau\,|y_{n+1}| & \text{anderenfalls.} \end{cases}$$

Verwenden Sie bei der Realisierung die Funktion `neville` aus Aufgabe 10.2.

Aufgabe 10.4*. Modifizieren Sie die Funktionen aus Aufgabe 9.6 und Aufgabe 10.3 so, dass die vollständingen Folgen $(y_1, \ldots, y_{n+1})$ der berechneten Approximationen zurückgegeben werden. Vergleichen Sie die Konvergenz des einseitigen Differenzenquotienten mit der Konvergenz der durch Richardson-Extrapolation gemäß Aufgabe 10.3 berechneten Approximationen an $f'(x)$. Schreiben Sie dazu ein Skript, dass folgende Aufgaben erfüllt:

- Für eine Funtkion $f \in \mathcal{C}^\infty(\mathbb{R})$ sollen Folgen der Differenzenquotienten mit und ohne Extrapolation berechnet werden.
- Es soll eine Tabelle der absoluten Fehler $|\Phi(h) - f'(x)|$ bzw. und relativen Fehler $|\Phi(h) - f'(x)|/|f'(x)|$ für beide Verfahren ausgegeben werden.
- Es soll ein (doppelt logarithmischer) Konvergenzgraph erstellt werden, in dem beide beide Verfahren direkt miteinander verglichen werden. Plotten Sie dazu die approximativen Werte y_n über $1/h_n$. Fügen Sie eine Legende, Achsenbeschriftungen und eine Überschrift ein.

Aufgabe 10.5. Man kann das Neville-Verfahren aus Aufgabe 10.2 so programmieren, dass zur Speicherung der Werte *keine* Matrix $(p_{j,m})_{j,m=1}^n$ aufgebaut wird, sondern die gegebenen y_j-Werte geeignet überschrieben werden. Dadurch wird kein weiterer Speicher benötigt. Man realisiere dieses Vorgehen in einer Funktion `neville2`.

Aufgabe 10.6. Bei der Richardson-Extrapolation des einseitigen Differenzenquotienten aus Aufgabe 10.3 kann man sich folgende Beobachtung zum Neville-Verfahren zu Nutze machen: Bei einer effizienten Implementierung des Neville-Verfahrens gemäß Aufgabe 10.2 überschreibt man den Vektor y durch die Diagonale $(p_{1,n}, p_{2,n-1}, \ldots, p_{n,1})$, und $p_{1,n}$ ist der gesuchte Wert. Fügt man nun einen weiteren Interpolationsknoten (x_{n+1}, y_{n+1}) hinzu, so muss man nicht noch einmal das vollständige Schema rechnen, sondern lediglich die „neue Diagonale“ $(p_{1,n+1}, p_{2,n}, \ldots, p_{n+1,1})$. Mit dieser Beobachtung muss man in jedem Schritt der Richardson-Extrapolation nur noch eine Schleife durchlaufen, nicht mehr zwei! Realisieren Sie dieses Vorgehen in einer Funktion `richardson2`.

Aufgabe 10.7. Der einseitige Differenzenquotient aus Aufgabe 9.6 konvergiert lediglich mit Ordnung $\alpha = 1$. Mit Hilfe der Taylorschen Formel kann man für $f \in \mathcal{C}^3(\mathbb{R})$ die Fehlerabschätzung

$$\Psi(h) - f'(x) = \mathcal{O}(h^2) \quad \text{mit} \quad \Psi(h) := \frac{f(x+h) - f(x-h)}{2h}$$

für den zentralen Differenzenquotienten $\Psi(h)$ beweisen. Schreiben Sie eine Funktion `diff2(f,x,h0,tau)`, die für $h_n := 2^{-n} h_0$ die Folge der zentralen Differenzenquotienten $\Psi(h_n)$ berechnet, bis gilt

$$|\Psi(h_n) - \Psi(h_{n+1})| \leq \begin{cases} \tau & \text{für } |\Psi(h_n)| \leq \tau, \\ \tau\,|\Psi(h_n)| & \text{anderenfalls.} \end{cases}$$

Die Funktion liefere in diesem Fall die vollständige Folge $(\Psi(h_0), \ldots, \Psi(h_n))$ der Iterierten zurück. Vergleichen Sie die Funktionen `diff` und `diff2` miteinander. Wie können Sie die mathematische Voraussage über das Konvergenzverhalten überprüfen?

Aufgabe 10.8. Modifizieren Sie Aufgabe 10.3, um den zentralen Differenzenquotienten zu extrapolieren, indem Sie (theoretisch!) das Interpolationspolynom p_n vom Grad $n-1$ zu den Punkten $(h_j^2, \Psi(h_j))$ für $j = 1, \ldots, n$ betrachten, d.h. $p_n(h^2) \approx \Psi(h)$, und dieses mit dem Neville-Verfahren bei $h = 0$ auswertet. Fügen Sie in den Plot aus Aufgabe 10.4 die Konvergenzgraphen für den zentralen Differenzenquotienten und die Richardson-Extrapolation des zentralen Differenzenquotienten ein.

Aufgabe 10.9. Schreiben Sie den Mergesort-Algorithmus aus Aufgabe 9.1–9.2 in *C* und erzeugen Sie mittels der MEX-Schnittstelle eine Matlab-Funktion `mergesort`, die ein Feld a aufsteigend sortiert und das sortierte Feld zurückgibt.Plotten Sie in einem doppellogarithmischen Plot die Laufzeiten der Matlab-Implementierung aus Aufgabe 9.1–9.2 und der C-MEX-Implementierung für Felder der Längen $N = 100 \cdot 2^n$ und $n = 0, 1, 2, \ldots$. Fügen Sie zum Verleich Steigungsgraden mit Steigung $\mathcal{O}(N)$ und $\mathcal{O}(N \log N)$ in den Plot ein.

Aufgabe 10.10. Eine weitere Quadraturregel ist die *summierten Trapezregel*. Hier berechnet man für gegebenes $n \in \mathbb{N}$ und $h := (b-a)/n$ die Approximation

$$I_n := \frac{h}{2}\Big(f(a) + 2\sum_{j=1}^{n-1} f(a+jh) + f(b)\Big). \tag{3}$$

Dies ist gerade das Integral über die stetige und stückweise affine Funktion p mit $p(a+jh) = f(a+jh)$. Man schreibe eine Funktion `trapezregel(f,a,b,tau)`, die die Folge der Approximationen I_n berechnet, bis gilt

$$|I_n - I_{n-1}| \leq \begin{cases} \tau & \text{für } |I_n| \leq \tau, \\ \tau\,|I_n| & \text{anderenfalls.} \end{cases}$$

In diesem Fall gebe man die vollständige Folge $(I_1, \ldots, I_n)$ der Approximationen zurück. Man teste die numerische Integration am Beispiel $f(x) = \exp(x)$ auf dem Intervall $[0, 10]$ und gebe abhängig von n den Fehler $|I - I_n|$ tabellarisch aus. Vergleichen Sie dies mit der summierten Rechteckregel aus Aufgabe 10.1.