Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Wintersemester 2010/11
13.Jänner 2010

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1*. Die sogenannte *Power-Iteration* approximiert (unter gewissen Voraussetzungen) den betragsgrößten Eigenwert $\lambda \in \mathbb{R}$ einer symmetrischen Matrix $A \in \mathbb{R}^{n\times n}$ sowie einen dazugehörigen Eigenvektor $x \in \mathbb{R}^n$. Dazu wählt man einen Startvektor $x^{(0)} \in \mathbb{R}^n \backslash \{0\}$, z.B. $x^{(0)} = (1, \ldots, 1) \in \mathbb{R}^n$. Dann definiert man induktiv für $k \in \mathbb{N}$ die Folgen

$$x^{(k)} := \frac{Ax^{(k-1)}}{\|Ax^{(k-1)}\|_2} \quad \text{und} \quad \lambda_k := x^{(k)} \cdot Ax^{(k)} := \sum_{j=1}^{n} x_j^{(k)} (Ax^{(k)})_j,$$

wobei $\|y\|_2 := \left(\sum_{j=1}^n y_j^2\right)^{1/2}$ die euklidische Norm bezeichne. Dann konvergiert die Folge (λ_k) gegen λ, und $(x^{(k)})$ konvergiert gegen einen Eigenvektor zu λ. Schreiben Sie eine Funktion `poweriteration`, die eine Matrix A, eine Toleranz $\tau > 0$ und optional auch einen Startvektor $x^{(0)}$ übernimmt, dann A auf Symmetrie überprüft und ggf. mit Fehlermeldung abbricht und schließlich die Folgen λ_k und $(x^{(k)})$ berechnet, bis gilt

$$\|Ax^{(k)} - \lambda_k x^{(k)}\|_2 \le \tau \quad \text{sowie} \quad |\lambda_{k-1} - \lambda_k| \le \begin{cases} \tau & \text{für } |\lambda_k| \le \tau, \\ \tau\,|\lambda_k| & \text{anderenfalls.} \end{cases}$$

Die Funktion liefere in diesem Fall λ_k und x_k zurück. Realisieren Sie die Funktion möglichst rechenökonomisch, d.h. vermeiden Sie unnötige Berechnungen (insb. von Matrix-Vektor-Produkten), indem Sie Ergebnisse ggf. zwischenspeichern. Sie können Ihre Funktion mit Hilfe der MATLAB-Funktion `eig` verifizieren. Verwenden Sie die Funktion `norm` sowie Matlab-Arithmetik, soweit wie möglich.

Aufgabe 11.2*. 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 $j = 1, \ldots, n$. 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 $p(t) = p_{1,n}$. Schreiben Sie 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

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

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. Speichern Sie den Source-Code unter `neville.m` in das Verzeichnis `serie11`. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 11.3*. Eine effiziente Implementierung des einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 8.3 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

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

Verwenden Sie bei der Realisierung die Funktion `neville` aus Aufgabe 11.1. Speichern Sie den Source-Code unter `richardson.m` in das Verzeichnis `serie11`.

Aufgabe 11.4*. Modifizieren Sie die Funktionen aus Aufgabe 8.3 und Aufgabe 11.3 so, dass die vollständingen Folgen $(y_1, \dots, y_{n+1})$ der berechneten Approximationen zurückgegeben werden. Vergleichen Sie die Konvergenz des einseitigen Differenzenquotienten mit der Konvergenz der durch Richardson-Extrapolation gemäß Aufgabe 11.3 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 11.5. Bei der Richardson-Extrapolation des einseitigen Differenzenquotienten aus Aufgabe 11.3 kann man sich folgende Beobachtung zum Neville-Verfahren zu Nutze machen: Bei einer effizienten Implementierung des Neville-Verfahrens gemäß Aufgabe 11.2 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 11.6. Der einseitige Differenzenquotient aus Aufgabe 8.3 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 11.7. Man schreibe eine rekursive Funktion `detlaplace`, die die Determinante $\det(A)$ einer Matrix $A \in \mathbb{R}^{n \times n}$ mit Hilfe des Laplaceschen Entwicklungssatzes berechnet. Sie können Ihre Funktion mit der Matlab-Funktion `det` verifizieren.

Aufgabe 11.8. Ein weiteres Verfahren zur Nullstellensuche ist das *Sekantenverfahren*. Dabei sind x_0 und x_1 gegebene Startwerte und man definiert induktiv x_{n+1} als Nullstelle der Geraden durch $(x_{n-1}, f(x_{n-1}))$ und $(x_n, f(x_n))$, d.h.

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

Verifizieren Sie diese Folge, und schreiben Sie eine Funktion `sekante(f,x0,x1,tau)` die die Folge der Iterierten berechnet, bis entweder

$$|f(x_n) - f(x_{n-1})| \leq \tau$$

oder

$$|f(x_n)| \leq \tau \quad \text{und} \quad |x_n - x_{n-1}| \leq \begin{cases} \tau & \text{für } |x_n| \leq \tau, \\ \tau |x_n| & \text{sonst} \end{cases}$$

gilt. In zweiten Fall werde $\widetilde{z} := x_n$ als Approximation der Nullstelle x_0 von f zurückgegeben. Im ersten Fall werde mit Fehlermeldung abgebrochen. — Neben x_n sollen zusätzlich die Folgen $(x_0, \ldots, x_n)$ der Nullstellen und der dazugehörigen Funktionswerte zurückgegeben werden.

Aufgabe 11.9. Implementieren Sie den Quicksort-Algorithmus, um einen Vektor $x \in \mathbb{R}^n$ zu sortieren: Quicksort wählt willkürlich ein Pivotelement aus der zu sortierenden Liste x, z.B. x_1. Dann zerlegt man die Liste in zwei Teillisten: $x^{(<)}$ enthält alle Elemente $< x_1$, $x^{(\geq)}$ enthält alle Elemente $\geq x_1$. Diese Teillisten werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Es besteht (in der Reihenfolge) aus der unteren Liste $x^{(<)}$, dem Pivotelement und der oberen Liste $x^{(\geq)}$. — Eine direkte Implementation dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, geht man wie folgt vor: Beginnend mit $j = 2$ sucht man ein Element $x_j \geq x_1$,

d.h. x_j gehört zu $x^{(\geq)}$. Ferner sucht man beginnend bei $k = n$ ein Element $x_k < x_1$, d.h. x_k gehört zu $x^{(<)}$. In diesem Fall vertauscht man x_j und x_k. Wenn sich die Zähler j und k treffen, liegt die Liste x bereits in der Form $(x^{(<)}, x_1, x^{(\geq)})$ vor. Es müssen nun nur noch die Teillisten sortiert werden.

Aufgabe 11.10. Das Integral $\int_a^b f\,dx$ einer stetigen Funktion f wird üblicherweise durch sogenannte Quadraturformeln berechnet. Die *Trapezregel* $I_1(f,a,b)$ sowie die *Simpson-Regel* $I_2(f,a,b)$ sind beispielsweise durch

$$I_1(f,a,b) := \frac{b-a}{2}\big(f(a)+f(b)\big) \quad \text{und} \quad I_2(f,a,b) := \frac{b-a}{6}\Big(f(a)+4\,f\big(\frac{a+b}{2}\big)+f(b)\Big)$$

gegeben. Schreiben Sie eine rekursive Funktion `adquad`, die das Integral $\int_a^b f\,dx$ mittels Simpson-Regel $I_2(f,a,b)$ und iterierter Intervallhalbierung

$$\int_a^b f\,dx = \int_a^{(a+b)/2} f\,dx + \int_{(a+b)/2}^b f\,dx \approx I_2(f,a,(a+b)/2) + I_2(f,(a+b)/2,b)$$

berechnet. Dabei soll der Integrationsbereich jeweils halbiert werden, falls

$$|I_1(f,a,b) - I_2(f,a,b)| > \varepsilon\,\max\{|I_1(f,a,b)|, |I_2(f,a,b)|\} \quad \text{und} \quad |b-a| > h_{\min}$$

ist. Der Funktion sollen neben f, a und b auch die Parameter ε und $h_{\min}$ übergeben werden. Sie können Ihren Code mit Hilfe der Matlab-Funktion `quad` verifizieren.