Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Wintersemester 2010/11
05. Jänner 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Aufgabe 10.1*. Das Δ^2-Verfahren von Aitken ist ein Verfahren zur Konvergenzbeschleunigung von Folgen. Für eine injektive Folge $(x_n)_{n\in\mathbb{N}}$ mit $\lim_{n\to\infty} x_n = x$ definiert man

$$y_n := x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}$$

Unter gewissen zusätzlichen Voraussetzungen an die Folge $(x_n)_{n\in\mathbb{N}}$ gilt dann

$$\lim_{n\to\infty} \frac{y_n - x}{x_n - x} = 0,$$

d.h. die Folge $(y_n)_{n\in\mathbb{N}}$ konvergiert schneller gegen x als $(x_n)_{n\in\mathbb{N}}$. Schreiben Sie eine Funktion `aitken`, die für einen Vektor $x \in \mathbb{R}^n$ mit Länge $n \geq 3$ den Vektor $y \in \mathbb{R}^{n-2}$ berechnet. Speichern Sie den Source-Code unter `aitken.m` in das Verzeichnis `serie10`.

Aufgabe 10.2*. Man kombiniere das Aitken-Verfahren aus Aufgabe 10.1 mit dem einseitigen Differenzenquotienten $\Phi(h)$ aus: Mit $h_n := 2^{-n}h_0$ betrachten wir die Folge der $x_n := \Phi(h_n)$ und erhalten daraus die Folge (y_n). Man schreibe eine Funktion `diffaitken`, die neben dem Function-Handle einer Funktion f den Auswertungspunkt x, die Schrittweite $h_0 > 0$ sowie die Toleranz $\tau > 0$ übernimmt und $y_{n+1} \approx f'(x)$ zurückliefert, sobald gilt

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

In jedem Schritt gebe man h, $|y_{n+1} - y_n|$ sowie y_{n+1} aus. Als Beispiel betrachte man die Berechnung von $e = \exp(1) = \exp'(1)$ und $\varepsilon = 10^{-12}$. Vergleichen Sie die Anzahl der Iterationen mit und ohne (d.h. $y_n = x_n$) Aitken-Verfahren. Speichern Sie den Source-Code unter `diffaitken.m` in das Verzeichnis `serie10`.

Aufgabe 10.3*. Gegeben seien eine Matrix $A \in \mathbb{R}^{n\times n}$ und eine rechte Seite $b \in \mathbb{R}^n$. Lösen Sie das Gleichungssystem $Ax = b$ mit dem *Gauß'schen Eliminationsverfahren*. Dies ist gerade das Vorgehen, wenn man ein lineares Gleichungssystem händisch löst:

- Zunächst bringt man die Matrix A auf obere Dreiecksform, indem man die Unbekannten eliminiert. Gleichzeitig modifiziert man die rechte Seite b.
- Das entstandene Gleichungssystem mit oberer Dreiecksmatrix A lässt sich dann einfach lösen.

Im ersten Eliminationsschritt zieht man geeignete Vielfache der ersten Zeile von den übrigen Zeilen ab und erhält dadurch eine Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

Im zweiten Eliminationsschritt zieht man nun geeignete Vielfache der zweiten Zeile von den übrigen Zeilen ab und erhält eine Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3} & \dots & a_{nn} \end{pmatrix},$$

Nach $n-1$ Eliminationsschritten erhält man also eine obere Dreiecksmatrix A. Berücksichtigen Sie, dass auch die rechte Seite $b \in \mathbb{R}^n$ geeignet modifiziert werden muss und machen Sie sich das Vorgehen zunächst an einem Beispiel mit $A \in \mathbb{R}^{2\times 2}$ sowie $A \in \mathbb{R}^{3\times 3}$ klar. Schreiben Sie eine Funktion `gauss`, die die Lösung von $Ax = b$ berechnet. Speichern Sie den Source-Code unter `gauss.m` in das Verzeichnis `serie10`.

Aufgabe 10.4*. Das Gauß'sche Eliminationsverfahren scheitert, falls im k-ten Schritt $a_{kk} = 0$ gilt, auch wenn das Gleichungssystem $Ax = b$ eine eindeutige Lösung x besitzt. Deshalb kann man das Verfahren um eine sogenannte *Pivot-Suche* erweitern:

- Im k-ten Schritt wählt man aus $a_{kk}, \dots, a_{nk}$ das betragsgrößte Element a_{pk}.
- Dann vertauscht man die k-te und die p-te Zeile von A (und b).
- Schließlich führt man den Eliminationsschritt aus wie zuvor.

Schreiben Sie eine Funktion `gausspivot`, die die Lösung von $Ax = b$ wie angegeben berechnet. (Man kann übrigens mathematisch beweisen, dass das Gauss-Verfahren mit Pivot-Suche genau dann durchführbar ist, wenn das Gleichungssystem $Ax = b$ eine eindeutige Lösung besitzt. Einen Beweis dazu sehen Sie in der Vorlesung zur Numerischen Mathematik.) Speichern Sie den Source-Code unter `gausspivot.m` in das Verzeichnis `serie10`.

Aufgabe 10.5. Wenn man im Gauß-Verfahren mit Pivot-Suche die Zeilen im Eliminationsschritt *wirklich* vertauscht (d.h. Speicher kopiert), führt dies auf unnötig viele Operationen und entsprechend lange Laufzeit des Programms. Es empfiehlt sich daher, die Vertauschung nur *virtuell* durchzuführen: Man startet mit einem Buchhaltervektor $\pi = (1, \dots, n)$. Im Vertauschungsschritt vertauscht man lediglich $\pi(p)$ mit $\pi(k)$. Im Source-Code sind jetzt die Zeilenindizes, d.h. der erste Index von a_{jk} sowie der Index von b_j geeignet zu modifizieren.

Aufgabe 10.6. Schreiben Sie eine Funktion `plotPotential(f)`, die den Plot eines Potentials $f = f(x, y)$ als farbige Projektion auf die Ebene erstellt. Zeichnen Sie 9 Konturlinien in den Plot. Geben Sie unter den Plot eine horizontale `colorbar`. Testen Sie Ihre Funktion mit dem Potential $f(x, y) = e^{-x^2-y^2}$. Speichern Sie den Source-Code unter `plotPotential.m` und das entstandene Bild als farbige Post-Script-Datei unter potential.eps in das Verzeichnis `serie10`.

Aufgabe 10.7. Schreiben Sie den Mergesort-Algorithmus aus Aufgabe 9.9–9.10 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 doppeltlogarithmischen Plot (`help loglog`) die Laufzeiten der Matlab-Implementierung aus Aufgabe 9.9–9.10 und der C-MEX-Implementierung für Felder der Längen $N = 100 \cdot 2^n$ und $n = 0, 1, 2, \dots$. Fügen Sie zum Verleich Steigungsgeraden mit Steigung $\mathcal{O}(N)$ und $\mathcal{O}(N \log N)$ in den Plot ein. Speichern Sie den Source-Code unter `mergesort.c` in das Verzeichnis `serie10`.

Aufgabe 10.8. Es sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische und positv definite Matrix, d.h. $Ax \cdot x > 0$ für $x \neq 0$. Dann existiert eine eindeutige untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit Diagonalelementen $\ell_{jj} > 0$ für alle

$j = 1 \ldots, n$ und $A = LL^T$, d.h. A besitzt eine spezielle LU-Zerlegung mit $U = L^T$. Man bezeichnet diese Faktorisierung als *Cholesky-Faktorisierung*. Leiten Sie aus der Formel für das Matrix-Matrix-Produkt $A = LL^T$ eine Formel für die Einträge von L her, und schreiben Sie eine Funktion `cholesky`, die L zurückliefert. Sie können Ihre Funktion in Matlab mittels `chol` verifizieren.

Aufgabe 10.9. 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 $m \geq 2$ und $j + m \leq n + 1$ 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

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

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 `serie10`. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 10.10. Man kann das Neville-Verfahren aus Aufgabe 10.9 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`.