Dirk Praetorius
Marcus Page

Sommersemester 2011
18. Mai 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 8

Aufgabe 8.1*. Die Frobeniusnorm einer Matrix $A \in \mathbb{R}^{m\times n}$ ist durch

$$\|A\|_F := \Big(\sum_{j=1}^{m}\sum_{k=1}^{n} A_{jk}^2\Big)^{1/2}$$

definiert. Schreiben Sie eine Funktion `frobeniusnorm`, die für eine gegebene Matrix A die Frobeniusnorm berechnet. Verwenden Sie nur elementare Arithmetik und geeignete Schleifen. Speichern Sie den Source-Code unter `frobenius.m` in das Verzeichnis `serie08`.

Aufgabe 8.2*. 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 $y \in \mathbb{R}^{n-2}$ mittels geeigneter Schleifen berechnet. Speichern Sie den Source-Code unter `aitken.m` in das Verzeichnis `serie08`.

Aufgabe 8.3*. Schreiben Sie eine alternative Funktion `aitken_vec`, die den Vektor $y \in \mathbb{R}^{n-2}$ aus Aufgabe 8.2 unter Vermeidung von Schleifen mittels geeigneter MATLAB-Arithmetik berechnet. Speichern Sie den Source-Code unter `aitken_vec.m` in das Verzeichnis `serie08`.

Aufgabe 8.4*. Schreiben Sie eine `void`-Funktion `pascal(k)`, die die ersten k-Stufen des Pascal'schen Dreiecks berechnet und als untere Dreiecksmatrix $P \in \mathbb{R}^{k\times k}$ zurückgibt: Jede Zeile dieses Schemas beginnt und endet mit 1. Die restlichen Zahlen werden als Summe nebeneinanderstehender Zahlen der vorherigen Zeile gebildet. Für $k = 5$ gilt beispielsweise

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

Realisieren Sie das Pascalsche Dreieck möglichst rechenökonomisch, insbesondere ohne die Darstellung der Einträge über den Binomialkoeffizienten.

Aufgabe 8.5. Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \dots, n, \quad k = i, \dots, n,$$

$$\ell_{ki} = \frac{1}{u_{ii}} \left(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \right) \quad \text{für } i = 1, \dots, n, \quad k = i+1, \dots, n,$$

$$\ell_{ii} = 1 \quad \text{für } i = 1, \dots, n,$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null. Schreiben Sie eine Funktion `computeLU`, die die LU-Zerlegung von A berechnet und zurückgibt. Dazu überlege man, in welcher Reihenfolge man die Einträge von L und U berechnen muss, damit die angegebenen Formeln wohldefiniert sind (d.h. alles was benötigt wird, ist bereits zuvor berechnet worden). Dabei sollen nur Schleifen und elementare Arithmetik verwendet werden. Sie können Ihrer Funktion mittels `lu` und `norm` in MATLAB verifizieren. Speichern Sie den Source-Code unter `computeLU.m` in das Verzeichnis `serie08`.

Aufgabe 8.6. Gegeben sei eine normalisierte untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit $\ell_{jj} = 1$ für alle $j = 1, \dots, n$. Zu gegebenem $y \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Lx = y$. Schreiben Sie eine Funktion `solveL`, die x berechnet. Dabei soll auf die offensichtlichen Nulleinträge von L nicht zugegriffen werden, um den Rechenaufwand gering zu halten. — Um den Algorithmus herzuleiten, schreibe man das Matrix-Vektor-Produkt $y = Lx$ komponentenweise für y_j mit $j = 1, \dots, n$ als Summe hin. Man überlege, wie die spezielle Gestalt von L die Laufindizes der Summe vereinfacht und löse diese Gleichung nach x_j auf. Dabei sollen nur Schleifen und elementare Arithmetik verwendet werden. Versuchen Sie, möglichst viele Schleifen durch Verwendung von Vektorarithmetik zu umgehen. Sie können Ihre Funktion mit Hilfe des MATLAB Backslash-Operators \ verifizieren. Speichern Sie den Source-Code unter `solveL.m` in das Verzeichnis `serie08`.

Aufgabe 8.7. Gegeben sei eine obere Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$ mit $u_{jj} \neq 0$ für alle $j = 1, \dots, n$. Für gegebenes $b \in \mathbb{R}^n$ existiert dann eine eindeutige Lösung $x \in \mathbb{R}^n$ von $Ux = b$. Schreiben Sie eine Funktion `solveU`, die die Lösung berechnet. Dabei sollen lediglich Schleifen und elementare Arithmetik verwendet werden. Versuchen Sie, möglichst viele Schleifen durch Verwendung von Vektorarithmetik zu umgehen. Sie können Ihre Funktion mit Hilfe des MATLAB Backslash-Operators \ verifizieren. Speichern Sie den Source-Code unter `solveU.m` in das Verzeichnis `serie08`.

Aufgabe 8.8. Schreiben Sie eine Funktion `solveLU`, die die Lösung x des linearen Gleichungssystems $Ax = b$ berechnet. Dabei soll die folgende Lösungsstrategie verwendet werden:

(1) Berechne die LU-Zerlegung von A.

(2) Löse $Ly = b$ nach y.

(3) Löse $Ux = y$ nach x.

Es gilt dann nämlich $Ax = LUx = Ly = b$. Verwenden Sie bei der Implementierung lediglich elementare Arithmetik sowie geeignete Schleifen. Den Speicheraufwand kann man hierbei (theoretisch) minimieren, indem man die Einträge der Matrix A geeignet durch die Einträge der Matrizen L und U überschreibt. Ferner kann man den Vektor b bei geeignetem Vorgehen, zunächst durch y und schliesslich durch x überschreiben. Dadurch wird insgesamt kein zusätzlicher Speicher benötigt. Realisieren Sie dieses Vorgehen, indem Sie Ihren Code aus den drei vorausgegangenen Aufgaben kombinieren und geeignet adaptieren. Sie können Ihre Funktion mit Hilfe des MATLAB Backslash-Operators \ verifizieren. Speichern Sie den Source-Code unter `solveLU.m` in das Verzeichnis `serie08`.

Aufgabe 8.9. Ändern Sie Ihre Funktion aus Aufgabe 8.1 dahingehend ab, dass sie ohne Schleifen auskommt und nur MATLAB-Vektorarithmetik verwendet. Speichern Sie den Source-Code unter `frobenius2.m` in das Verzeichnis `serie08`.

Aufgabe 8.10. Das Integral $\int_a^b dx$ einer stetigen Funktion $f : [a, b] \to \mathbb{R}$ kann man durch eine sogenannte Quadraturformel

$$\int_a^b f\,dx \approx \sum_{j=1}^{n} \omega_j f(x_j)$$

approximieren, wobei man sich einen Vektor $x \in \mathbb{R}^n$ mit $x_1 < \cdots < x_n$ vorgibt und die Funktion f (formal = theoretisch) durch ein Polynom $p(x) = \sum_{j=1}^{n} a_j x^{j-1}$ vom Grad $\leq n-1$ mit $p(x_j) = f(x_j)$ approximiert. Die Gewichte ω_j lassen sich aus der Forderung berechnen, dass

$$\int_a^b q\,dx = \sum_{j=1}^{n} \omega_j q(x_j) \quad \text{für alle Polynome } q \text{ vom Grad} \leq n-1$$

gilt. Dies ist nämlich äquivalent zur Lösung des linearen Gleichungssystems

$$\frac{b^{k+1}}{k+1} - \frac{a^{k+1}}{k+1} = \int_a^b x^k\,dx = \sum_{j=1}^{n} \omega_j x_j^k \quad \text{für alle } k = 0, \ldots, n-1.$$

Warum ist das so? Schreiben Sie eine Funktion `integrate`, die f und den Vektor $x \in \mathbb{R}^n$ übernimmt und den approximativen Wert des Integrals zurückgibt. Dazu bauen Sie das lineare Gleichungssystem möglichst effizient auf und lösen dieses mittels Backslash-Operator. Mit Hilfe des Ergebnisvektors $\omega \in \mathbb{R}^n$ ergibt sich das approximative Integral als Skalarprodukt mit dem $f(x)$-Vektor.