Dirk Praetorius
Winfried Auzinger

Sommersemester 2018
11.04.2018

Übungen zur Vorlesung
Computermathematik

Serie 3

Aufgabe 3.1. Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt strikt diagonal dominant falls

$$\sum_{\substack{k=1\\k\neq j}}^{n} |A_{jk}| < |A_{jj}| \quad \text{für alle } j \in \{1,\dots,n\}.$$

Ist eine Matrix symmetrisch, strikt diagonal domininant und es gilt $A_{jj} > 0$ für alle $j \in \{1,\dots,n\}$ so ist A positiv definit. Schreiben Sie eine Funktion `constructSPDmatrix` die für eine gegebene Dimension n eine zufällige symmtrische und positiv definite Matrix $A \in \mathbb{R}^{n\times n}$ erzeugt. **Hinweis:** Verwenden Sie `rand()`. Für alle $A \in \mathbb{R}^{n\times n}$ ist $A * A^T$ symmetrisch. Überlegen Sie wie Sie daraus geeignete SPD-Matritzen bauen können.

Aufgabe 3.2. Gegeben sei eine symmetrische und postiv definite (SPD) Matrix $A \in \mathbb{R}^{n\times n}$ und ein Vektor $b \in \mathbb{R}^n$. Das CG-Verfahren bietet eine effizente Möglichkeit das Gleichungssystem $Ax = b$ zu lösen. Schreiben Sie eine Funktion `cgsolve`, die eine Matrix $A \in \mathbb{R}^{n\times n}$, eine rechte Seite $b \in \mathbb{R}^n$ sowie eine Toleranz $\tau > 0$ übernimmt und die Lösung des obigen Gleichunssystems berechnet. Gehen Sie dazu folgendermaßen vor: Für einen beliebigen Startvektor $x^{(0)} \in \mathbb{R}^n$ berechnet man $r^{(0)} := b - Ax^{(0)}$ und $d^{(0)} := r^{(0)}$. Man defniert induktiv für $k \in \mathbb{N}$ die Folgen:

$$\alpha^{(k)} := \frac{r^{(k)} \cdot r^{(k)}}{d^{(k)} \cdot Ad^{(k)}}, \quad x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)}, \quad r^{(k+1)} := r^{(k)} - \alpha^{(k)} Ad^{(k)}$$

sowie

$$d^{(k+1)} := r^{(k+1)} + \frac{r^{(k+1)} \cdot r^{(k+1)}}{r^{(k)} \cdot r^{(k)}} d^{(k)}.$$

Die CG-Iteration soll abbrechen falls $|r^{(k+1)}| \leq \tau$ und anschließend $x^{(k+1)} \approx x$ zurückgeben. Testen Sie Ihre Funktion mit geeigneten Beispielen. Was beobachten sich für die Anzahl der CG-Schritte für $\tau \to 0$ und große Dimensionen $n \in \mathbb{N}$. **Hinweis:** Verwenden Sie Aufgabe 3.1.

Aufgabe 3.3. Das Δ^2-Verfahren von Aitken ist ein Verfahren zur Konvergenzbeschleunigung von Folgen. Für eine injektive Folge (x_n) mit $x = \lim_{n\to\infty} x_n$ definiert man

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

Unter gewissen Voraussetzungen an die Folge (x_n) gilt dann

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

d.h. die Folge (y_n) konvergiert schneller gegen x als (x_n). Schreiben Sie eine MATLAB-Funktion `aitken`, die einen Vektor $x \in \mathbb{R}^N$ übernimmt und den Vektor $y \in \mathbb{R}^{N-2}$ zurückgibt. Verwenden Sie dazu geeignete Schleifen. Schreiben Sie eine alternative MATLAB-Funktion `aitken_vec`, die den Vektor $y \in \mathbb{R}^{N-2}$ mittels geeigneter Vektor-Arithmetik statt Schleifen berechnet. Überlegen Sie sich, wie Sie ihren Code auf Korrektheit testen können! Was passiert für eine geometrische Folge $x_n := q^n$ mit $0 < q < 1$?

Aufgabe 3.4. Schreiben Sie eine Funktion `diffaitken`, die die Ableitung einer Funktion in einem Punkt x approximiert. Dazu verwende man den einseitigen und den zentralen Differenzenquotienten

$$\Phi(h) = \frac{f(x+h) - f(x)}{h} \quad \text{bzw.} \quad \Phi(h) = \frac{f(x+h) - f(x-h)}{2h}.$$

Für gegebene Startschrittweite $h_0 > 0$ berechne man jeweils die Folgen $h_n := 2^{-(n-1)}h_0$, $x_n := \Phi(h_n)$. Berechnen Sie weiters den Wert der Extrapolierten ϕ_n. Setzen Sie dazu $\phi_n := x_n$ für $n = 1, 2$, und $\phi_n := y_{n-2}$ für $n \geq 3$, wobei $y_i N$ die Folge der Atiken-Extrapolation aus Aufgabe 3.3 bezeichnet. Berechnen Sie die experimentielle Konvergenzrate für den einseiteigen und zentralen Differenzenquotienten mit und ohne Aitken-Extrapolation. Visualisieren Sie ihre Ergebnisse. Welche Raten beobachten Sie?

Aufgabe 3.5. 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 MATLAB-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. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 3.6. Man kann das *Neville-Verfahren* aus Aufgabe 3.5 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 MATLAB-Funktion `neville2`.

Aufgabe 3.7. Eine effiziente Implementierung des einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 3.4 verwendet die vorherigen Werte $\Phi(h_0), \ldots, \Phi(h_n)$, indem man (theoretisch!) das Interpolationspolynom p_n vom Grad $n-1$ zu den Punkten $(h_j, \Phi(h_j))$ für $j = 1, \ldots, n$ 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 $y_n := p_n(0)$. Schreiben Sie 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 3.5.

Aufgabe 3.8. 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. $x^{(0)} = (1,\dots,1) \in \mathbb{R}^n$. Man definiert 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 (in einem geeigneten Sinn) gegen einen Eigenvektor zu λ. Schreiben Sie eine Funktion `poweriteration`, die eine Matrix A, eine Toleranz τ und 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

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

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.