Winfried Auzinger
Dirk Praetorius

Sommersemester 2013
13. März 2013

Übungen zur Vorlesung Computermathematik

Serie 2

Aufgabe 2.1. 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.

Aufgabe 2.2. Schreiben Sie eine alternative MATLAB-Funktion `aitken_vec`, die den Vektor $y \in \mathbb{R}^{N-2}$ aus Aufgabe 2.1 mittels geeigneter Vektor-Arithmetik statt Schleifen berechnet.

Aufgabe 2.3. Erweitern Sie den Code von Folie 66, indem Sie das Konvergenzverhalten des einseitigen und zentralen Differenzenquotientens mit dem Aitkenschen Δ^2-Verfahren verbessern. Welche Konvergenzraten beobachten Sie? Visualisieren Sie diese geeignet!

Aufgabe 2.4. Schreiben Sie eine Funktion `diffaitken`, die die Ableitung einer Funktion in einem Punkt x approximiert. Dazu verwende man den zentralen Differenzenquotienten

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

Für gegebene Startschrittweite $h_0 > 0$ und Toleranz τ berechne man die Folgen $h_n := 2^{-(n-1)}h_0$, $x_n := \Phi(h_n)$, und $\phi_n := x_n$ für $n = 1, 2$ bzw. sei $\phi_n := y_{n-2}$ für $n \geq 3$ der Wert der Aitken-Extrapolierten. Die Iteration werde beendet, falls $n \geq 2$ und

$$|\phi_n - \phi_{n-1}| \leq \begin{cases} \tau & \text{falls } |\phi_n| \leq \tau, \\ \tau|\phi_n| & \text{sonst.} \end{cases}$$

In diesem Fall werde ϕ_n als Approximation der Ableitung zurückgegeben.

Aufgabe 2.5. 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. $x^{(0)} = (1, \ldots, 1) \in \mathbb{R}^n$. 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

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

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.

Aufgabe 2.6. 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 2.7. Man kann das Neville-Verfahren aus Aufgabe 2.6 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 2.8. Das Integral $\int_a^b f\,dx$ 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 $p(x_j) = f(x_j)$ für alle $j = 1, \ldots, n$ approximiert und das Integral $\int_a^b p\,dx$ exakt berechnet. 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 `quadrature`, die zu gegebenem Spaltenvektor $x \in \mathbb{R}^n$ und Integrationsgrenzen $a < b$ den Zeilenvektor der Gewichte $\omega \in \mathbb{R}^n$ 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 in der Form `omega*fx`, wenn `fx` der Spaltenvektor der Funktionsauswertungen ist. Visualisieren Sie das Konvergenzverhalten des Fehlers

$$e_n = \Big| \int_a^b f\,dx - \texttt{omega} * \texttt{fx} \Big|$$

über der Länge n eines äquidistanten x-Vektors (d.h. $x_j = a + (b-a)(j-1)/(n-1)$) für die Funktionen $f(x) = \exp(x)$ und $f(x) = x^{2.5}$ über dem Integrationsbereich $[0, 10]$.