Dirk Praetorius
Winfried Auzinger

Sommersemester 2017
22.03.2017

Übungen zur Vorlesung
Computermathematik

Serie 3

Aufgabe 3.1. 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. 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 3.2. 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 3.1 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 3.3. 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

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

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

Aufgabe 3.4. Man kann das *Neville-Verfahren* aus Aufgabe 3.3 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 3.5. Eine effiziente Implementierung des einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 3.2 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 $y_n := p_n(0)$. 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

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

Verwenden Sie bei der Realisierung die Funktion `neville` aus Aufgabe 3.3.

Aufgabe 3.6. 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

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

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

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

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.

Aufgabe 3.7. Sei $f : [a, b] \to \mathbb{R}$ eine stetige Funktion. Für $N \in \mathbb{N}$ und $x_j := a + j\,(b-a)/N$ mit $j = 0, \ldots, N$, definieren wir die *zusammengesetzte Mittelpunktsregel*

$$I_N := \frac{b-a}{N} \sum_{j=1}^{N} f\big((x_{j-1} + x_j)/2\big). \tag{1}$$

Da I_N eine Riemannsumme ist, wissen wir, dass

$$\lim_{N\to\infty} I_N = \int_a^b f\,dx.$$

Für $f \in C^2[a, b]$, kann man sogar zeigen, dass

$$\Big| \int_a^b f\,dx - I_N \Big| = \mathcal{O}(N^{-2}). \tag{2}$$

Schreiben Sie eine MATLAB Funktion

```
int = midpointrule(f,n,a,b),
```

welche für die Folge $N = 2^k$ und $k = 0, \ldots, n$, den Vektor `int` der entsprechenden Werte I_N berechnet und zurückgibt. Überlegen Sie sich, wie Sie ihren Code auf Korrektheit testen können! Verifizieren Sie experimentell die in (2) angegebene Konvergenzordnung. **Hinweis:** Testen Sie die Quadratur mit Polynome von verschiedenem Grad. Berechnen Sie das Ergebnis auch analytisch. Was fällt Ihnen dabei auf?

Aufgabe 3.8. Modifizieren Sie die Funktion `midpointrule` aus Aufgabe 3.7 folgendermaßen.

- Wird `midpointrule(f,n)` ohne Intervallgrenzen a, b aufgerufen, so wird $\int_{-1}^{1} f\ dx$ berechnet.
- Der Aufruf `midpointrule(f,n,a,b)` soll wie in Aufgabe 3.4 den Vektor $I_N \approx \int_a^b f\ dx$ zurückliefern. Beachten Sie, im Fall $b < a$ gilt $\int_a^b f\ dx = -\int_b^a f\ dx$. Geben Sie in diesem Fall zusätzlich eine Warnung aus.
- Im Falle `midpointrule(f,n,a,b,'nodes')` soll zusätzlich zum Vektor `int` auch der Vektor `nodes` der Stützstellen x_j mit $j = 0, \ldots, 2^n$ zurück gegeben werden.