Winfried Auzinger
Gregor Gantner
Alexander Haberl
Dirk Praetorius

Sommersemester 2016
16. März 2016

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. 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 alternative MATLAB-Funktion `aitken_vec`, die den Vektor $y \in \mathbb{R}^{N-2}$ aus Aufgabe 3.1 mittels geeigneter Vektor-Arithmetik statt Schleifen berechnet.

Aufgabe 3.3. 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. Überlegen Sie sich, wie Sie ihren Code auf Korrektheit testen können! Was sind geeignete Test-Beispiele?

Aufgabe 3.4. 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

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

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! **Hinweis:** Testen Sie die Quadratur mit Polynome von verschiedenem Grad. Berechnen Sie das Ergebnis auch analytisch. Was fällt Ihnen dabei auf?

Aufgabe 3.5. Modifizieren Sie die Funktion `midpointrule` aus Aufgabe 3.4 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.

Aufgabe 3.6. Alternativ zum Bisektionsverfahren aus der Vorlesung kann man zur Berechnung einer Nullstelle einer Funktion $f : [a, b] \to \mathbb{R}$ das *Newton-Verfahren* verwenden. Ausgehend von einem Startwert x_0 definiert man induktiv eine Folge (x_n) wie folgt: Zu gegebenem x_k sei x_{k+1} die Nullstelle der Tangente an den Graphen von f im Punkt $(x_k, f(x_k))$, d.h. $x = x_{k+1}$ erüllt $0 = f(x_k) + f'(x_k)(x - x_k)$. Auflösen nach x zeigt

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Realisieren Sie das Newton-Verfahren in einer Funktion `newton(f,fprime,x0,tau)`, wobei die Iteration abgebrochen wird, falls entweder

$$|f'(x_n)| \le \tau$$

oder

$$|f(x_n)| \leq \tau \quad \text{und} \quad |x_n - x_{n-1}| \leq \begin{cases} \tau & \text{für } |x_n| \leq \tau, \\ \tau|x_n| & \text{sonst} \end{cases}$$

gilt. In jedem Fall werde x_n als Approximation der gesuchten Nullstelle zurückgeben, wobei im ersten Fall zusätzlich eine Warnung ausgegeben werden soll.– Neben x_n sollen die Folgen $(x_0, \ldots, x_n)$ der approximativen Nullstellen und der dazugehörigen Funktionswerte zurückgegeben werden. Testen Sie Ihre Implementierung mit der Funktion $f(x) = x^2 + e^x - 2$.

Aufgabe 3.7. Überlegen Sie sich mindestens drei weitere nicht-triviale Beispiele um Ihre Implementierung des *Newton-Verfahrens* zu testen. Schreiben Sie weiters eine MALAB-Funktion `testnewton(f,fprime,x0,tau)` um Ihre Lösung optisch zu verifizieren. Plotten Sie dazu Ihre Testfunktion $f(x)$ und Ihre Approximation der Nullstelle. Achten Sie dabei auf geeignete Skalierung im Plot, um Ihre Lösung möglichst gut überprüfen zu können! **Hinweis:** Sie können `scatter` verwenden um einzelne Punkte zu plotten.

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