Marcus Page
Dirk Praetorius

Sommersemester 2011
25. Mai 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1*. 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 $j = 1, \dots, n$. 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 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. Speichern Sie den Source-Code unter `neville.m` in das Verzeichnis `serie09`. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 9.2*. Man kann das Neville-Verfahren aus Aufgabe 9.1 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 Funktion `neville2`. Speichern Sie den Source-Code unter `neville2.m` in das Verzeichnis `serie09`.

Aufgabe 9.3*. Schreiben Sie eine Funktion `power`, die für gegebene reelle Zahlen $x > 1$ und $C > 0$ die kleinste Zahl $n \in \mathbb{N}$ berechnet mit $x^n > C$. Dabei soll die Funktion `log` nicht verwendet werden. Speichern Sie den Source-Code unter `power` in das Verzeichnis `serie09`.

Aufgabe 9.4*. Die Sinus-Funktion hat die Reihendarstellung

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Wir betrachten die Partialsummen

$$S_n(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Schreiben Sie eine Funktion `sinreihe`, die für gegebene $x \in \mathbb{R}$ und $\varepsilon > 0$ den Wert $S_n(x)$ zurückliefert, sobald

$$|S_n(x) - S_{n-1}(x)|/|S_n(x)| \leq \varepsilon \quad \text{oder} \quad |S_n(x)| \leq \varepsilon$$

gilt. Schreiben Sie die Funktion so, dass diese mit einer Schleife auskommt und dass x^{2k+1} und $(2k+1)!$ möglichst kostensparend realisiert werden. Man verwende also insbesondere keine (vor- oder selbst implementierten) Funktionen zur Berechnung der Potenz oder der Faktoriellen. Speichern Sie den Source-Code unter `sinreihe.m` in das Verzeichnis `serie09`.

Aufgabe 9.5. Gegeben sei eine stetige Funktion $f : [a, b] \to \mathbb{R}$. Es gelte

$$f(a) \cdot f(b) \leq 0.$$

Dann hat f eine Nullstelle z_0 in $[a, b]$, die im Folgenden mittels Bisektion (= Intervallhalbierung) approximiert werden soll: Der Bisektionsalgorithmus arbeitet wie folgt: In jedem Bisektionsschritt definiert man $c := (a + b)/2$ als Intervallmittelpunkt. Aufgrund der Voraussetzung gilt

$$f(a) \cdot f(c) \leq 0 \quad \text{oder} \quad f(c) \cdot f(b) \leq 0.$$

Im Fall $f(a) \cdot f(c) \leq 0$ liegt eine Nullstelle im Intervall $[a, c]$, und man ersetzt daher b durch c. Im Fall $f(c) \cdot f(b) \leq 0$ liegt eine Nullstelle im Intervall $[c, b]$, und man ersetzt daher a durch c. In beiden Fällen geht man also vom Intervall $[a, b]$ zu einem Teilintervall halber Länge über. Schreiben Sie eine Funktion `bisection`, die als Parameter f, a, b und eine Toleranz $\tau > 0$ übernimmt. Dabei werde f als `function handle` übergeben und ausgewertet. Der Bisektionsschritt soll so lange wiederholt werden, bis man vom Startintervall $[a, b]$ zu einem Intervall $[a, b]$ mit Länge $|b - a| \leq 2\,\tau$ übergegangen ist. In diesem Fall gebe man $(a+b)/2$ zurück. Es gilt dann $|(a+b)/2 - z_0| \leq |a-b|/2 \leq \tau$, d.h. $(a+b)/2$ ist eine Approximation einer Nullstelle z_0 bis auf eine Genauigkeit von τ. Speichern Sie den Source-Code unter `bisection.m` in das Verzeichnis `serie09`. Was ist der Unterschied, wenn man die Funktion f in der Form `'...'` übergibt und mittels `feval` auswertet? Kann `feval` auch für ein `function handle` verwendet werden?

Aufgabe 9.6. Eine Variante zur Berechnung einer Nullstelle einer Funktion $f : [a, b] \to \mathbb{R}$ ist das *Newton-Verfahren*. Ausgehend von einem Startwert x_0 definiert man induktiv eine Folge (x_n) durch

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

Die anschauliche Idee dahinter ist, dass man an die Funktion f in x_k die Tangente legt. Das nächste Folgenglied ist dann die Nullstelle x_{k+1} dieser Tangente. Leiten Sie die gegebene Formel mathematisch her. Realisieren sie das Newton-Verfahren in einer Funktion `newton`, die als Parameter f, f', x_0 und eine Toleranz $\tau > 0$ übernimmt. Die Iteration werde abgebrochen, falls entweder

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

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. Im ersten Fall breche man mit Fehlermeldung ab, siehe `help error`. Speichern Sie den Source-Code unter `newton.m` in das Verzeichnis `serie09`.

Aufgabe 9.7. Für $x > 0$ konvergiert die Folge

$$x_1 := \frac{1}{2}(1+x), \quad x_{n+1} := \frac{1}{2}\Big(x_n + \frac{x}{x_n}\Big) \quad \text{für } n \geq 1$$

gegen $\sqrt{x}$. Schreiben Sie eine Funktion `wurzel`, die für gegebene $x > 0$ und $\tau > 0$ als Ergebnis das erste Folgenglied $y = x_n$ zurückgibt, für das gilt

$$\frac{|x_n - x_{n+1}|}{|x_n|} \leq \tau \quad \text{oder} \quad |x_n| \leq \tau.$$

Speichern Sie den Source-Code unter `wurzel.m` in das Verzeichnis `serie09`. Welcher Zusammenhang besteht mit dem Newton-Verfahren aus Aufgabe 9.6?

Aufgabe 9.8. Schreiben Sie eine Funktion `sqrt2`, die für gegebene $x > 0$ und $\tau > 0$ die Wurzel $\sqrt{x}$ mit Hilfe des Bisektionsverfahrens aus Aufgabe 9.5 approximiert, sodass der approximative Wert a die Fehlerschranke $|a - \sqrt{x}| \leq \tau$ erfüllt: Welche Funktion $f : [0, \infty) \to \mathbb{R}$ wählt man? Wie wählt man das Startintervall $[a, b]$? Vergleichen Sie die Anzahl der Iterationen mit der Anzahl der Iterationen in Aufgabe 9.7 und geben Sie jeweils tabellarisch die Fehler $|\sqrt{2} - x_n|$ für beide Verfahren aus. Speichern Sie den Source-Code unter `sqrt2.m` in das Verzeichnis `serie09`.

Aufgabe 9.9. 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 $x \in \mathbb{R}^n \backslash \{0\}$, d.h. $Ax = \lambda x$. Dazu wählt man einen Startvektor $x^{(0)} \in \mathbb{R}^n \backslash \{0\}$, z.B. $x^{(0)} = (1, \ldots, 1) \in \mathbb{R}^n$. Dann definiert man 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 $\|y\|_2 := \big(\sum_{j=1}^n y_j^2\big)^{1/2}$ die euklidische Norm bezeichne. Dann konvergiert die Folge (λ_k) gegen λ, und $(x^{(k)})$ konvergiert gegen einen Eigenvektor zu λ. Schreiben Sie eine Funktion `poweriteration`, die eine Matrix A und eine Toleranz $\tau > 0$ übernimmt, dann A auf Symmetrie überprüft und ggf. mit Fehlermeldung abbricht und schließlich die Folgen λ_k und $(x^{(k)})$ berechnet, bis gilt

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

Die Funktion liefere in diesem Fall λ_k und x_k zurück. Sie können Ihre Funktion mit Hilfe der MATLAB-Funktion `eig` verifizieren.

Aufgabe 9.10. Modifizieren Sie den Code aus Aufgabe 9.9 so, dass optional auch ein beliebiger Startvektor übergeben werden kann, siehe `help varargin`. Realisieren Sie die Funktion ferner möglichst rechenökonomisch, d.h. vermeiden Sie unnötige Berechnungen (insb. von Matrix-Vektor-Produkten), indem Sie Ergebnisse ggf. zwischenspeichern. Verwenden Sie die Funktion `norm` sowie Matlab-Arithmetik, soweit wie möglich.