Dirk Praetorius
Marcus Page

Sommersemester 2011
01. Juni 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Aufgabe 10.1*. Für einen stetigen Integranden $f : [a, b] \to \mathbb{R}$ berechnet man das Integral $I := \int_a^b f\,dx$ numerisch über geeignete Summen. Bei der *summierten Trapezregel* berechnet man für gegebenes $N \in \mathbb{N}$ und $h := (b-a)/N$

$$I_N := \frac{h}{2}\Big(f(a) + 2\sum_{j=1}^{N-1} f(a+jh) + f(b)\Big).$$

Es gilt dann im Limes

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

Schreiben Sie eine Funktion

$$\texttt{I} = \texttt{quadrature}(\texttt{f}, \texttt{a}, \texttt{b}, \texttt{N})$$

die für ein gegebenes Function-Handle `f`, Integrationsgrenzen $a < b \in \mathbb{R}$ und eine Ordnung $N \in \mathbb{N}$ die summierte Trapezregel

$$I_N := \frac{h}{2}\Big(f(a) + 2\sum_{j=1}^{N-1} f(a+jh) + f(b)\Big), \quad \text{mit} \quad h := (b-a)/N$$

mittels geeigneter Schleifen berechnet und zurückgibt. Speichern Sie den Source-Code unter `quadrature1.m` in das Verzeichnis `serie10`.

Aufgabe 10.2*. Schreiben Sie einen MATLAB-Code für

$$\texttt{I} = \texttt{quadrature}(\texttt{f}, \texttt{a}, \texttt{b}, \texttt{N})$$

der die summierte Trapezregel realisiert, wobei *alle* auftretenden Schleifen durch *Vektorarithmetik* eliminiert werden. Gehen Sie davon aus, dass die Implementierung (d.h. das Function-Handle) `f` der Funktion f einen Vektor `x` übernimmt und als Ergebnis die Funktionswerte als Vektor `y` gleicher Dimension zurückgibt, d.h. $y_j = f(x_j)$ lässt sich simultan berechnen. Testen Sie Ihren Code mit verschiedenen Funktionen `f`. Speichern Sie den Source-Code unter `quadrature2.m` in das Verzeichnis `serie10`.

Aufgabe 10.3*. Man schreibe eine Funktion

$$\texttt{I} = \texttt{integrate}(\texttt{f}, \texttt{a}, \texttt{b}, \texttt{eps})$$

die für $N = 2^n$ und $n = 0, 1, 2, 3, \ldots$ eine Approximation $I = I_{2N} \approx \int_a^b f\,dx$ zurückliefert. Dabei soll $N = 2^n$ minimal sein mit der Eigenschaft

$$|I_{2N} - I_N| \le \texttt{eps} \cdot \max\{|I_{2N}|, |I_N|\}.$$

Ferner sollen folgende Anforderungen berücksichtigt werden:

- Gehen Sie davon aus, dass die Implementierung `f` der Funktion f einen Vektor `x` übernimmt und als Ergebnis die Funktionswerte als Vektor `fx` gleicher Dimension zurückgibt.
- Beachten Sie, dass alle Auswertungen von f zur Berechnung von I_N auch zur Berechnung von I_{2N} benötigt werden. Schreiben Sie die Funktion möglichst rechenökonomisch, d.h. vermeiden Sie unnötige Funktionswertungen von f. Insbesondere soll *nicht* die Funktion `quadrature` verwendet werden.
- Schreiben Sie die Funktion ferner möglichst speicherökonomisch, d.h. vermeiden Sie —soweit als möglich— die Speicherung der berechneten Werte I_N sowie das Anlegen von Hilfsspeicher.

Aufgabe 10.4*. Unter einer **rekursiven Funktion** versteht man eine Funktion, die sich selber aufruft, zusammen mit einer Abbruchbedingung. Das folgende Beispiel ist Ihnen allen bekannt: Für $n \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}$ definieren wir $f(0) = 1$ und $f(n) := n \cdot f(n-1)$ für $n \geq 1$. Um welche Funktion handelt es sich? Man schreibe sich dazu beispielsweise alle Faktoren von $f(5)$ hin: $f(5) = 5 \cdot f(4) = 5 \cdot 4 \cdot f(3) \ldots$. Implementieren Sie diese rekursive Funktion in einem Programm, das $n \in \mathbb{N}_0$ einliest und $f(n)$ ausgibt. Wählen Sie einen geeigneten Namen für den Source-Code und speichern Sie ihn ins Verzeichnis `serie10`.

Aufgabe 10.5. Die Fibonacci-Folge, definiert durch $x_0 := 0$, $x_1 := 1$ und $x_{n+1} = x_n + x_{n-1}$ haben Sie in der Vorlesung bereits kennengelernt. Eine rekursive Implementierung, die zu gegebenem n den Wert x_n berechnet haben Sie ebenfalls in der Vorlesung gesehen. Schreiben Sie nun eine nicht rekursive Funktion `fibonacci`, die dasselbe leistet, wobei die Berechnung aber über geeignete Schleifen realisiert wird. Welche der beiden Funktionen ist effizienter und warum? Speichern Sie den Source-Code unter `fibonacci.m` in das Verzeichnis `serie10`.

Aufgabe 10.6. In den Aufgabe 8.2 und 8.3 haben Sie das Aitken'sche Δ^2-Verfahren kennengelernt und implementiert. Dieses Verfahren soll nun auf die summierte Trapezregel angewendet werden. Schreiben Sie eine Funktion

$$\texttt{I} = \texttt{aitken}(\texttt{f}, \texttt{a}, \texttt{b}, \texttt{eps})$$

wobei Sie das Δ^2-Verfahren auf die Folge

$$x_n := I_N = \frac{h}{2}\Big(f(a) + 2\sum_{j=1}^{N-1} f(a+jh) + f(b)\Big)$$

(aus den obigen Aufgaben) mit $N = 2^n$ und $h := (b-a)/N$ anwenden. Berechnen Sie die Folgenglieder y_n der Δ^2-Folge, bis erstmalig

$$|y_{n+1} - y_n| \leq \texttt{eps} \cdot \max\{|y_n|, |y_{n+1}|\}.$$

gilt. In diesem Fall werde $I := y_{n+1} \approx \int_a^b f\,dx$ zurückgegeben. Ferner sollen folgende Anforderungen berücksichtigt werden:

- Gehen Sie davon aus, dass die Implementierung `f` der Funktion f einen Vektor `x` übernimmt und als Ergebnis die Funktionswerte als Vektor `fx` gleicher Dimension zurückgibt.
- Schreiben Sie die Funktion möglichst rechenökonomisch, d.h. vermeiden Sie unnötige Funktionswertungen von f (also `quadrature` *nicht* verwenden!)
- Schreiben Sie die Funktion möglichst speicherökonomisch, d.h. speichern Sie *nicht* die gesamten Vektoren x_n und y_n, sondern lediglich die benötigten Folgenglieder.

Aufgabe 10.7. Es sei $L \in \mathbb{R}^{n\times n}$ eine untere Dreiecksmatrix mit $\ell_{jj} \neq 0$ für alle $j = 1, \ldots, n$. Dann ist L invertierbar, und die Inverse lässt sich wie folgt rekursiv berechnen: Wir schreiben L in Block-Form

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}$$

mit $L_{11} \in \mathbb{R}^{p\times p}$, $L_{21} \in \mathbb{R}^{q\times p}$ und $L_{22} \in \mathbb{R}^{q\times q}$, wobei $n = p + q$ gilt. Man beachte, dass L_{11} und L_{22} wieder untere Dreiecksmatrizen sind mit $\ell_{jj} \neq 0$. Üblicherweise wählt man $p = n/2$, falls n gerade ist, bzw. $p = (n-1)/2$, falls n ungerade ist. Offensichtlich wird die Inverse L^{-1} dann durch

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} L_{21} L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}$$

gegeben. Schreiben Sie eine Funktion `invertL`, die die Inverse L^{-1} rekursiv berechnet. Speichern Sie den Source-Code unter `invertL.m` in das Verzeichnis `serie10`. Sie können die Korrektheit Ihrer Funktion mit Hilfe der Matlab-Funktion `inv` überprüfen.

Aufgabe 10.8. Sie haben bisher sowohl kompilierte (C), als auch interpretierte (MATLAB) Programmiersprachen kennengelernt. Beschreiben Sie, worin sich diese beiden Varianten unterscheiden. Wo liegen jeweils Vor- und Nachteile. Einige Programmiersprachen (z.B. Java) verwenden eine so genannte *Virtual Machine* und gewinnen dadurch *Plattformunabhängigkeit*. Wie funktioniert das? Was sind hierbei Vor- und Nachteile?

Aufgabe 10.9. Schreiben Sie eine rekursive Funktion `detlaplace`, die die Determinante $\det(A)$ einer Matrix $A \in \mathbb{R}^{n\times n}$ mit Hilfe des Laplaceschen Entwicklungssatzes berechnet. Speichern Sie den Source-Code unter `detlaplace.m` in das Verzeichnis `serie10`. Sie können Ihre Funktion mit der MATLAB-Funktion `det` verifizieren.

Aufgabe 10.10. Die Quotientenfolge $(a_{n+1}/a_n)_{n\in\mathbb{N}}$ zur Fibonacci-Folge $(a_n)_{n\in\mathbb{N}}$,

$$a_0 := 1, \quad a_1 := 1, \quad a_n := a_{n-1} + a_{n-2} \quad \text{für } n \geq 2,$$

konvergiert gegen den goldenen Schnitt $(1 + \sqrt{5})/2$. Insbesondere konvergiert die Differenz

$$b_n := \frac{a_{n+1}}{a_n} - \frac{a_n}{a_{n-1}}$$

gegen Null. Schreiben Sie eine Funktion `cauchy`, die zu gegebenem $k \in \mathbb{N}$ die kleinste Zahl $n \in \mathbb{N}$ mit $|b_n| \leq 1/k$ zurückgibt. Speichern Sie den Source-Code unter `cauchy.m` in das Verzeichnis `serie10`.