Winfried Auzinger
Dirk Praetorius
Michele Ruggeri

Sommersemester 2014
19. März 2014

# Übungen zur Vorlesung Computermathematik

## Serie 3

**Aufgabe 3.1.** Aitken's $\Delta^2$-process is a series acceleration method, used for improving the rate of convergence of a sequence. Given an injective sequence $(x_n)$ such that the limit $x = \lim_{n\to\infty} x_n$ exists, consider the sequence $(y_n)$ defined by

$$y_n := x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n} \quad \text{for all } n \geq 0. \tag{1}$$

Under certain assumptions on the sequence $(x_n)$, it holds

$$\lim_{n\to\infty} \frac{x - y_n}{x - x_n} = 0,$$

i.e., $(y_n)$ converges towards $x$ and the convergence is faster than that of $(x_n)$. Write a MATLAB function `aitken`, which, given a vector $x \in \mathbb{R}^N$, returns the corresponding Aitken vector $y \in \mathbb{R}^{N-2}$. Use appropriate loops.

**Aufgabe 3.2.** Write a MATLAB function `aitken_vec`, which, given a vector $x \in \mathbb{R}^N$, returns the corresponding Aitken vector $y \in \mathbb{R}^{N-2}$ from Aufgabe 3.1. Avoid loops, and use only appropriate vector/matrix functions and arithmetics instead.

**Aufgabe 3.3.** Extend the code of slide 78 by adding the $\Delta^2$-method from Aufgabe 3.1 (or Aufgabe 3.2) to improve the convergence behavior of the central and forward difference quotients. Which are the observed convergence rates? Visualize them appropriately.

**Aufgabe 3.4.** Write a MATLAB function `diffaitken`, which computes the approximation of the derivative of a function $f$ in a point $x$ through the central difference quotient

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

Given the function $f$, the point $x$, an initial parameter $h_0 > 0$ and a tolerance $\tau > 0$, the function returns an approximation of the derivative obtained as follows: For $n \geq 1$, compute $h_n := 2^{-(n-1)} h_0$, $x_n := \Phi(h_n)$, and $\phi_n$ defined by

$$\phi_n := \begin{cases} x_n & \text{if } n = 1, 2, \\ y_{n-2} & \text{if } n \geq 3, \end{cases}$$

where, for $n \geq 3$, we apply the $\Delta^2$-method from Aufgabe 3.1–3.2 (define $y_{n-2}$ through (1)). The iteration stops when $n \geq 2$ and

$$|\phi_n - \phi_{n-1}| \leq \begin{cases} \tau & \text{if } |\phi_n| \leq \tau, \\ \tau|\phi_n| & \text{else,} \end{cases}$$

and the function returns $\phi_n$ as approximation of the derivative.

**Aufgabe 3.5.** One possible algorithm for eigenvalue computations is the *Power Iteration*. It approximates (under certain assumptions) the eigenvalue $\lambda \in \mathbb{R}$ with the greatest absolute value of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ as well as the corresponding eigenvector $x \in \mathbb{R}^n$. The algorithm is obtained as follows: Given a vector $x^{(0)} \in \mathbb{R}^n\backslash\{0\}$, e.g., $x^{(0)} = (1,\dots,1) \in \mathbb{R}^n$, define the sequences

$$x^{(k)} := \frac{Ax^{(k-1)}}{\|Ax^{(k-1)}\|_2} \quad \text{and} \quad \lambda_k := x^{(k)} \cdot Ax^{(k)} := \sum_{j=1}^{n} x_j^{(k)} (Ax^{(k)})_j \quad \text{for } k \in \mathbb{N},$$

where $\|y\|_2 := \left(\sum_{j=1}^n y_j^2\right)^{1/2}$ denotes the Euclidean norm. Then, under certain assumptions, $(\lambda_k)$ converges towards $\lambda$, and $(x^{(k)})$ converges towards an eigenvector associated to $\lambda$ (in an appropriate sense). Write a MATLAB function `poweriteration`, which, given a matrix $A$, a tolerance $\tau$ and an initial vector $x^{(0)}$, verifies whether the matrix $A$ is symmetric. If this is not the case, then the function displays an error message and terminates (use `error`). Otherwise, it computes $(\lambda_k)$ and $(x^{(k)})$ until

$$\|Ax^{(k)} - \lambda_k x^{(k)}\|_2 \le \tau \quad \text{and} \quad |\lambda_{k-1} - \lambda_k| \le \begin{cases} \tau & \text{if } |\lambda_k| \le \tau, \\ \tau|\lambda_k| & \text{else,} \end{cases}$$

and returns $\lambda_k$ and $x^{(k)}$. Realize the function in an efficient way, i.e., avoid unnecessary computations (especially of matrix-vector products) and storage of data. Then, compare `poweriteration` with the built-in MATLAB function `eig`. Use the function `norm`, as well as arithmetics.

**Aufgabe 3.6.** Let $U \in \mathbb{C}^{n\times n}$ be an upper triangular matrix, i.e., $u_{jk} = 0$ for $j > k$, such that $u_{jj} \neq 0$ for all $j = 1,\dots,n$. Given $b \in \mathbb{C}^n$, there exists a unique solution $x \in \mathbb{C}^n$ of $Ux = b$ (Why?). Write a MATLAB function `solveU`, which, given an upper triangular matrix $U$ as above and a vector $b \in \mathbb{C}^n$, computes the unique solution $x \in \mathbb{C}^n$ of $Ux = b$. Use only loops (but try to avoid them by suitable vector/matrix computations) and arithmetics. You must not use the MATLAB backslash operator to solve the linear system, but you can use it to test your implementation.

**Aufgabe 3.7.** Write a MATLAB function `mergesort` and an auxiliary function `merge` (included in the file `mergesort.m`) with the following features:
- Given two row vectors with entries sorted into ascending order $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$, the function `merge` merges them (through a suitable loop) to obtain a row vector $c \in \mathbb{R}^{m+n}$ sorted into ascending order, e.g., for $a = (1,3,3,4,7)$ and $b = (1,2,3,8)$ the function returns $c = (1,1,2,3,3,3,4,7,8)$. The function exploits the fact that the vectors $a$ and $b$ are already sorted, and therefore must not include any sorting algorithm (or the MATLAB function `sort`).
- Given a row vector $c \in \mathbb{R}^N$, the recursive function `mergesort` returns it sorted into ascending order. The algorithm should be implemented as follows: If $N \le 2$, then $c$ is manually sorted. If $N > 2$, $c$ is halved into two parts, $a$ and $b$, which are recursively sorted by calling `mergesort` to sort $a$ and $b$, and `merge` to merge the sorted vectors $a$ and $b$ to build the sorted vector $c$.

**Aufgabe 3.8.** Write a MATLAB script, which visualizes the runtime of `mergesort` for random vectors $x \in \mathbb{R}^N$ and $N = 100 \cdot 2^n$ with $n = 0,1,2,\dots$. Devise suitable plots to visualize the computational cost of your implementation. What is your expectation for a vector of length $2^N$?