Winfried Auzinger
Dirk Praetorius
Michele Ruggeri

Sommersemester 2014
12. März 2014

Übungen zur Vorlesung Computermathematik

Serie 2

Aufgabe 2.1. Write a MATLAB function `pnorm`, which, given a vector $x \in \mathbb{C}^n$ and $1 \le p < \infty$, returns the ℓ^p-norm of x

$$\|x\|_p := \Big(\sum_{j=1}^n |x_j|^p\Big)^{1/p}.$$

Avoid loops, and use only arithmetics and appropriate vector/matrix functions and indexing instead.

Aufgabe 2.2. Write a MATLAB function `cut`, which, given a vector $x \in \mathbb{R}^N$ and a constant $S \ge 0$, returns a vector $y \in \mathbb{R}^n$, obtained from x by removing all the entries x_j with $|x_j| > S$, e.g., $x = (0,2,1,4,5,0,0,1,2) \in \mathbb{R}^9$ and $S = 1$ yield $y = (0,1,0,0,1) \in \mathbb{R}^5$. Avoid loops, and use only arithmetics and appropriate vector/matrix functions and indexing instead.

Aufgabe 2.3. Write a MATLAB function `matrix`, which, given $n \in \mathbb{N}$, returns the matrix $A \in \mathbb{N}^{n\times n}$ with $A_{jk} = j + k$ and the checkerboard matrix $B \in \mathbb{N}^{n\times n}$ with

$$B_{jk} = \begin{cases} 1 & \text{for } j+k \text{ even,} \\ 0 & \text{for } j+k \text{ odd.} \end{cases}$$

Generate B from A exploiting `mod`. Avoid loops, and use only arithmetics and appropriate vector/matrix functions and indexing instead.

Aufgabe 2.4. Any polynomial of degree n is uniquely determined by the $n+1$ values of its coefficients. Consider the polynomials $p(x) = \sum_{j=0}^m a_j x^j$ and $q(x) = \sum_{k=0}^n b_k x^k$, as well as the vectors of their respective coefficients $a \in \mathbb{C}^{m+1}$ and $b \in \mathbb{C}^{n+1}$. The sum $p+q$ is a polynomial of degree $\max\{m,n\}$. Write a MATLAB function `addpol`, which, given $a \in \mathbb{C}^{m+1}$ and $b \in \mathbb{C}^{n+1}$, returns the coefficient vector $c \in \mathbb{C}^{\max\{m,n\}}$ of $(p+q)(x) = \sum_{\ell=0}^{\max\{m,n\}} c_\ell x^\ell$. The function has to work for both column and row input vectors a and b, but it always returns a column vector c. To this end, the function `reshape` or the syntax `vector(:)` might be useful. Avoid loops and `if`-conditions, and use only appropriate arithmetics and vector/matrix functions and indexing instead.

Aufgabe 2.5. Write a MATLAB function `diffpol` which, given a polynomial $p(x) = \sum_{j=0}^n a_j x^j$ defined through the coefficient vector $a \in \mathbb{C}^{n+1}$, returns the coefficient vector of the first derivative p'. The function has to work for both column and row input vectors, but it always returns a column vector. Avoid loops, and use only appropriate arithmetics and vector/matrix functions and indexing instead.

Aufgabe 2.6. Write a MATLAB function `evalpol` which, given a polynomial $p(x) = \sum_{j=0}^{n} a_j x^j$ defined through its coefficient vector $a \in \mathbb{C}^{n+1}$, and a matrix $x = (x_{jk}) \in \mathbb{C}^{M \times N}$ of evaluation points, returns the evaluation matrix $\big(p(x_{jk})\big) \in \mathbb{C}^{M \times N}$. The function has to work for both column and row input vectors. Do not forget that all the quantities are possibly complex-valued. Avoid loops, and use only appropriate arithmetics and vector/matrix functions and indexing instead. (**Hint:** Use the function `reshape` to simplify the problem, e.g., to deal with vectors instead of matrices.)

Aufgabe 2.7. The integral $\int_a^b f\,dx$ of a continuous function $f : [a, b] \to \mathbb{R}$ can be approximated as a weighted sum of function values at specified points within the domain of integration by using a so-called quadrature formula of the form

$$\int_a^b f\,dx \approx \sum_{j=1}^{n} \omega_j f(x_j).$$

Given a vector of quadrature points $x \in \mathbb{R}^n$ with $a \le x_1 < \cdots < x_n \le b$, such a formula might be obtained by approximating the function f through a polynomial $p(x) = \sum_{j=1}^{n} a_j x^{j-1}$ of degree $\le n-1$ which satisfies $p(x_j) = f(x_j)$ for all $j = 1, \ldots, n$. Then, the weights ω_j can be derived from the condition

$$\int_a^b q\,dx = \sum_{j=1}^{n} \omega_j q(x_j) \quad \text{for all polynomials } q \text{ of degree} \le n-1.$$

This is actually equivalent to the solution of the linear system

$$\frac{b^{k+1}}{k+1} - \frac{a^{k+1}}{k+1} = \int_a^b x^k\,dx = \sum_{j=1}^{n} \omega_j x_j^k \quad \text{for all } k = 0, \ldots, n-1.$$

Why? Write a MATLAB function `integrate`, which, given the column vector $x \in \mathbb{R}^n$ of quadrature points, returns the corresponding row vector $\omega \in \mathbb{R}^n$ of weights. To this end, build the linear system of equations in an efficient way and solve it by using the backslash operator. Avoid loops, and use only appropriate arithmetics and vector/matrix functions and indexing instead. (**Remark:** With the vector $\omega \in \mathbb{R}^n$, it is possible to compute the approximated integral by the matrix product with the $f(x)$-column vector.)

Aufgabe 2.8. Consider a lower triangular matrix $L \in \mathbb{R}^{n \times n}$ such that all the diagonal entries are non-zero, i.e., $\ell_{jj} \neq 0$ for all $j = 1, \ldots, n$. Then, L has the form

$$L = \begin{pmatrix} \ell_{11} & 0 & \cdots & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \ell_{n-1,1} & \ell_{n-1,2} & \cdots & \ell_{n-1,n-1} & 0 \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{n,n-1} & \ell_{nn} \end{pmatrix}.$$

Since $\det(L) = \prod_{j=1}^{n} \ell_{jj} \neq 0$, L is invertible, and the inverse might be recursively computed as follows: Write L in block form as

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

with $L_1 1 \in \mathbb{R}^{p\times p}$, $L_{21} \in \mathbb{R}^{q\times p}$ and $L_{22} \in \mathbb{R}^{q\times q}$, where $p + q = n$. Standard choices for p (and consequently q) are $p = n/2$ for even n and $p = (n-1)/2$ for odd n. Note that L_{11} and L_{22} are still invertible lower triangular matrices. Straightforward calculations show that the inverse has the following block form

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

Write a MATLAB function `invertL`, which, given an invertible lower triangular matrix $L \in \mathbb{R}^{n\times n}$, computes the inverse L^{-1} according to the aforementioned recursive procedure. The correctness of the implementation can be checked by use of `inv`. Avoid loops, and use only appropriate arithmetics and vector/matrix functions and indexing instead. (**Remark:** The recursion goes down to $n = 2$, where the inverse is explicitly given by the aforementioned formula.)