Sheet R00 - Introduction to Matlab

This is a voluntary exercise sheet dedicated to get a basic knowledge of Matlab. No points will be given, and no correction will be shown in the exercise.

In this lecture you will have the chance to learn many interesting theoretical as well as practical topics. In any case you have a problem understanding, please always feel free to write to the mailing list. This should be a place where you students can talk freely about the lecture, so do not hesitate to ask *and* reply! Of course the tutors are on the mailing list as well and will reply as well.

Assignment 1) Introduction to Matlab *(0Pts)*

Read the following chapters of the Matlab documentation. (Depending on you Matlab version the chapters may be named and ordered different)

- Matlab → Mathematics → Matrices and Linear Algebra → Function Summary
- Matlab → Programming → Data Structures →
  - Creating and Concatenating Matrices
  - Accessing Elements of a Matrix
  - Getting Information about a Matrix
  - Resizing and Reshaping Matrices
  - Multidimensional Arrays

Assignment 2) Matlab Commands *(0Pts)*

- Write a Matlab script containing the symbol, command or an example for each of the following exercises. For the tasks 1 to 18 you can get 0.5 credits for each task. For task 19 and 20 you can get up to 4 credits and for task 21 you can get up to 2 credits.

a) Create a comment.

b) Create a 1×4 row vector.

c) Create a 5×1 column vector.

d) Create a zero matrix using a function provided by Matlab.

e) Print the second row of a 4×3 matrix.

f) Print the third column of a 4×4 matrix.

g) Transpose a matrix.

h) Create two matrices of equal size $(\mathrm{m} \times \mathrm{m})$. Multiply them once using conventional matrix multiplication and once using element-wise multiplication.

i) Concatenate two matrices vertically, as well as horizontally using a Matlab command.

j) Print the size of a matrix.

k) Change the structure of a 8 $\times$ 7 matrix to 14 $\times$ 4 using a Matlab command.

l) Replicate a 3 $\times$ 1 vector to a matrix of size 3 $\times$ 1000.

m) Replace all matrix entries less than 0 by 0.

n) Clear all active variables.

o) Clear all active variables, functions and breakpoints.

p) Create a vector containing numbers from 1 to 100 with a gap of 7 between the numbers.

q) Create a vector with 100 entries. Set every second element to 0.

r) Create a vector with 100 entries. Delete every second element.

s) Create 2 matrices of size 1000×3 containing random numbers. You can interpret the rows of such a matrix as vectors of size 1×3. Compute the dot product of those vectors using loops. This means you have to iterate over the rows of the 1000 $\times$ 3 matrix and compute the dot product between the vectors represented by the current matrix row. Now, try to compute the dot product without using loops. Compare the runtimes of your implementation (Hint: To measure the runtime, you can use the `tic`, `toc` command described in the Matlab Documentation.). Did you recognize something, while comparing the times (loops vs. no loops)?

t) The following scenario is given. We want to invert 1000 2 $\times$ 2 matrices. The 2 $\times$ 2 matrices are

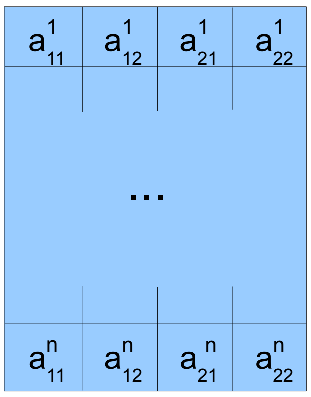

Figure 1: Memory layout of the 2 $\times$ 2 matrices

represented by a row (see Figure 1). The numbers written as subscripts represent the position of the original matrix entry of the 2 $\times$ 2 matrices. The superscript denotes the current matrix $n \in [1 \ldots 1000]$. Now, create a matrix with a size of 1000 $\times$ 4 using the Matlab command `rand`. Every row in that matrix represents a 2 $\times$ 2 matrix (compare Figure 1). Note, that the memory layout of the 2 $\times$ 2 matrices must not be changed. That means we don't want to change the current structure of the 2 $\times$ 2 matrices represented as a row. Don't change the 1 $\times$ 4 matrices to 2 $\times$ 2 matrices and don't change the *inv* command. We can now use Cramer's Rule to compute the inverse of our 1000, as row represented, 2 $\times$ 2 matrices. Compute the inverses of the created 2 $\times$ 2 matrices without using any loops.

u) Write a Matlab function, which accepts 2 $m \times m$ matrices. The function should be able to compute the sum as well as the product of both matrices depending on the number of return parameters.