Prof. Dr. Thomas Schultz
Olivier Morelle (morelle@uni-bonn.de)

B-IT
Bonn-Aachen International
Center for Information Technology
January 10, 2024

Winter term 2024/25

Computer Science for Life Scientists

Assignment Sheet 12

Solution has to be uploaded by January 17, 2025, 10:00 a.m., via eCampus

- This exercise can be submitted in **small groups** of 2-3 students. Submit each solution only once, but clearly indicate who contributed to it by forming a team in eCampus. Remember that all team members have to be able to explain all answers.
- Remember to include proper **documentation** in all your code, in particular, docstrings for functions.
- Please submit your answers as a single file in .ipynb or .pdf format.

If you have questions concerning the exercises, please use the forum on eCampus.

Exercise 1 (Binary Conversion and Arithmetics, *5 Points*)

In this exercise, you will develop your skills in converting between binary and decimal as well as floating-point numbers, and in performing binary arithmetic operations.

a) Convert the decimal number 72 to its 8-bit binary representation. (.5P)
b) Convert the binary number 11010010 to its decimal representation. (.5P)
c) Convert the floating-point number 4.125 to its binary representation. (1P)
d) Convert the binary number 10101.11 to its floating-point representation. (1P)
e) Add the binary numbers 10101101 and 11011. Show the steps of the addition and the final result in binary form. (1P)
f) Subtract the binary numbers 111011 and 1010. Show the steps of the subtraction and the final result in binary form. (1P)

Exercise 2 (Round-off Errors in Summation, *5 Points*)

In order to study round-off errors in Python, create a list of length $2^{10} + 1$. The first item should be 1, followed by 2^{10} items with value 2^{-53} each.

- Compute the sum of the items front-to-back and print the result, with the full accuracy provided by Python.
- Compute the sum of the items back-to-front and print the result, with the full accuracy provided by Python.
- Compute the sum of the items using `numpy.sum` and print the result, with the full accuracy provided by Python.

Please explain why these three approaches led to different results. Which of the methods produced the most accurate result in this specific case? Do you think that method should be preferred in general? Why? Does the difference persist when replacing all copies of 2^{-53} with 2^{-52} in the input? Why?

Exercise 3 (Solving Underdetermined Systems Using QR Factorization, *5 Points*)

Let $Ax = b$ be a system of linear equations defined by a matrix $A \in \mathbb{R}^{m \times n}$ of full rank and a right-hand side $b \in \mathbb{R}^m$. In the lecture, we saw how we can use the QR factorization to find the unique solution $x \in \mathbb{R}^n$ if A is square ($m = n$), and a solution in the least squares sense if $m > n$. In this exercise, we will consider the underdetermined case with $m < n$.

Consider a QR factorization of A^T, i.e., $Q \in \mathbb{R}^{n \times n}$ is orthogonal and $R \in \mathbb{R}^{n \times m}$ an upper triangular matrix such that $A^T = QR$. Also consider the submatrix $R_1 \in \mathbb{R}^{m \times m}$ so that $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$. Let $y_1 \in \mathbb{R}^m$ satisfy $R_1^T y_1 = b$ and $y_2 \in \mathbb{R}^{n-m}$ be an arbitary vector. Argue why any

$$x = Q \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

is a solution of the underdetermined system of linear equations $Ax = b$. (2P) Propose an algorithm to calculate one particular solution x from given Q, R, and b (2P). Do we require the reduced or the full QR decomposition? (1P)

Exercise 4 (Gram-Schmidt Algorithm for QR Factorization, *10 Points*)

Hint: For the following tasks, you may use the numpy library to represent vectors and matrices, compute dot products and norms, etc.

a) Write the following functions to compute the reduced QR factorization of a full-rank matrix A. The functions must return matrices Q and R.
   - CGS(A), using the *classical Gram-Schmidt* algorithm (3P)
   - MGS(A), using the *modified Gram-Schmidt* algorithm (1P)

   Hint: Remember that Python and numpy start indexing at 0, while the lecture slides assume indexing starts at 1.

b) Write a function evaluate(A,Q,R) to compute the error of a QR factorization. Use the following measures as the factorization and orthogonality error: (2P)
   - $E_{qr} = \|A - QR\|_2$
   - $E_o = \left\|Q^T Q - I\right\|_2$, where I is the identity matrix

c) Define $A = \begin{bmatrix} 0.70000 & 0.70711 \\ 0.70001 & 0.70711 \end{bmatrix}$ as a numpy array with single precision floating point format. Use your functions from part a) to compute the QR factorization of A. Evaluate the accuracy of your factorization using the measures in part b). Repeat the same process, by defining A with double precision floating point format. In which case do you get a more accurate factorization? (1P) Why does it not make a difference if you use CGS or MGS in this case ? (1P) Provide a different example in which MGS produces a more accurate result than CGS. (2P)

Good Luck!