Course 02635, October 27th, 2021.

s194119 – Martin Ægidius

Group members:

s194120, Frederik Gade

s194146, Nikoline Mai Bøgely Rehn

s190566, Oliver Zacho

Assignment 1: forward substitution

Function structure: The program is designed as a nested for-loop, which calculates the sum $\sum_{i=1}^{k-1}(b_i R_{ik})$ for each value of i and followingly updates the k'th solutional element in b with $b_k = \left(b_k - \sum_{i=1}^{k-1}(b_i R_{ik})\right)/(\alpha + R_{diag_k})$, with k also indexing the k'th diagonal element of the matrix. The sum is reset to zero on every iteration of k. Using row major storage order for storing the matrix, the k'th diagonal element can be accessed in a for-loop with $(k = 0; k < n)$ using $R_{diag_k} = R[k + n \cdot k]$, as the first diagonal element is located in $R_{0,0}$. This method is only valid for square matrices. The program returns value $\{-1,0\}$ whether it fails/succeeds.

The program checks the input for different errors: **1.** If pointers point to elements. **2.** If $\alpha \in \{NaN, \infty\}$. **3.** If input-pointer contains any values $b_k \in \{NaN, \infty\}$. **4.** If n exists, and has a valid value. **5.** If R contains invalid values $R_{ij} \in \{NaN, \infty\}$. **6.** Checking if there is a unique solution to the system of equations, by ensuring $\alpha + R_{ii} \neq 0$ for all $i = 1, \dots, n$. If this is untrue, the program returns -1, as proceeding also would lead to undefined behavior in the program (division by 0). **7.** If $\boldsymbol{R}$ is an upper triangle matrix, which is a necessity for using backwards substitution. **8.** Detecting overflow and underflow by checking if $b_k \in \{NaN, \infty\}$ ever evaluates to true. Sadly, there is no way of checking whether the dimensions of $\boldsymbol{R}$ and $\boldsymbol{b}$ match, if $\boldsymbol{R}$ is a squared matrix, and if the matrix actually square with $n \times n$, in C. This would require a different input than a pointer.

Unimplemented numerical considerations: as b_k is updated using subtraction, loss of precision due to cancellation will occur. It would be easy to implement a check whether $b[k] - \sum_{i=1}^{k-1}(b_i R_{ik}) \in [-\sigma; \sigma]$ to a certain tolerance σ, and display a warning as cancellation will be worst when the numbers are close. Special cases where the fraction $f > DBL\_MAX \rightarrow f = \infty$, this returns -1 – even though a solution actually exists. The issue also arises when $f < DBL\_MIN \rightarrow f = 0$. Thus, some valid equations sadly will return -1. In general, the program will be prone to failure when the magnitudes of numbers are very large/small.

Assignment 2: solving the triangular Sylvester equation

Function structure: The program takes two pointers to array2d_t-structs. After input-checks described below, a double-nested loop is initiated with the structure $k\{i\{j\}\}$. The j-loop is used for calculating $\sum_{j=1}^{k-1} \boldsymbol{c_j} R_{jk}$. The ij-loop is used for looping columnwise through C in order to update each element of row k with the expression $\boldsymbol{c_k} = \boldsymbol{c_k} - \sum_{j=1}^{k-1} c_j R_{jk}$, as $\boldsymbol{c_k}$ actually is $\{c_{k,1}, \dots, c_{k,n}\}$ in matrix $\boldsymbol{C}$. In the beginning of the i-loop, the sum is reset, which is necessary as it is a function of k. The k-loop contains the second step of solving the system, which is equivalent to a forward-substitution (earlier described) using $\alpha_k = R_{kk}$. The substitution is only made for $\boldsymbol{c_k}$, and thus only runs one row at a time by pushing the start-element of the pointer C by $+n \cdot k$ and using the next n elements. Function outputs $\{-2, -1, 0\}$ are mapped to $\{\text{invalid input}, \text{num. error}, \text{success}\}$.

The program has input-validation: **1.** Do pointers to the structs exist. **2.** does n have a valid value. **3.** Are the matrices square and both of size $n \times n$. **4.** Is $\boldsymbol{R}$ an upper-triangle matrix. **5.** are both structs stored using RowMajor. **6.** Are any $c_{ij} \in \{NaN, \infty\}$, indicating earlier numerical errors/rounding. Likewise for r_{ij}. If any of these checks are true, the program returns -2. **7.** All checks in assignment 1 are carried out. This ensures, that numerical errors in the updated $\boldsymbol{C}$ return -1. In case that forwardsub fails, the Sylvester-solver outputs -1.

Unimplemented numerical considerations: The program presents a numerical error if sumk > $DBL\_MAX \rightarrow sumk = \infty$. In order to avoid this, the sum should be implemented more elegantly than using $+=$ (e.g. adding negative and positive terms pairwise, starting with smallest numbers), as the program will fail even though a solution may exist. Also, precision is lost when $sumk$ is very large/small, due to floating point precision. Using the $+=$ operation should be avoided due to error accumulation (instead e.g. Kahan summation could be used).

Testing: Forward substitution is tested using mock-input. We know which values are expected, e.g.:

$$R = \begin{bmatrix} 1 & 3 \\ 0 & 4 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, n = 2, \alpha = 3 \rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \cdot (3+1)^{-1} \\ \left(5 - (x_1 \cdot 3)\right) \cdot (3+4)^{-1} \end{pmatrix} = \begin{pmatrix} 0.75 \\ 0.39 \end{pmatrix}$$

Which is compliant with the program-output when printing results. Testing the code with incompatible matrix-vector-dimensions returns 0 and results in undefined behavior which yields a wrong result. This is considered the *user's responsibility*. Both programs are tested with invalid entries (non-triangular, wrong dimensions, non-square, NaN and ∞ values, unspecific solutions) and designed matrixes expected prone to numerical issues. Cross-testing has been carried out with Maple, yielding consistent results. CodeJudge has been the final test for both programs.