Course number: 02635,
Date: October 21, 2020

August Semrau Andersen, s183918
(Coded with Karl Ulbæk, s183931)

Mathematical Software Programming: Assignment 1

Part I: Forward substitution

Function/program structure: how did you organize your code?
The function consists mainly of one for loop that runs from 0 to n-1. Each loop, we first check that $\alpha + R[k][k]$ is not zero, we then calculate the sum $\sum_{i=1}^{k-1} b_i R_{ik}$, and lastly we overwrite b[k] with the solution x[k]. If successful, we finally return 0.

Did you consider any numerical aspects in your implementation?
We avoid division by zero by checking, as mentioned above, that $\alpha + R[k][k]$ is not 0.

How did you test your code? What tests did you perform to ensure correctness?
We wrote a script, fwdsub_test.c, which dynamically allocates R and b and inputs these into fwdsub(). We have checked that the resulting b-vector is correct according to own calculation, and that it return -1 for non-legal division.

Part II: Triangular Sylvester equation

Function/program structure: how did you organize your code?
The unchanged function fwdsub is initiated in the top of the script and its code is pasted at the buttom. Another way to do it would have been a separate script and inclusion in the header file, but that was not ideal for codejudge here.
First step of tri_sylvester_solve() is checking that the input is valid, ie. pointers are not NULL-pointers. The dimensions are further checked, as the number of rows and columns have to be equal for both R and C and across these two as well.
Hereafter, solving the Sylvester equation starts:
We loop over all rows in C, each loop we temporarily store the row $\mathbf{c}_k$ as 'c_k' using the vector structure given in the header file matrix.h.
For each row $\mathbf{c}_k$ we first want to calculate $\mathbf{c}_k - \sum_{i=1}^{k-1} \mathbf{c}_j R_{jk}$, which we do element-wise. Alternatively, we could define a separate vector holding the sum and then subtract it from c_k.
In the same loop iteration, we also perform the second operation $\mathbf{c}_k \leftarrow \left(R_{kk} I + R^T\right)^{-1} \mathbf{c}_k$. This is done using the function fwdsub from part one, which unchanged fulfill the correct computation.
Lastly we overwrite the given k'th row of C with the one we have just newly computed.
After iterating over all rows, we return 0 and finish.

Did you consider any numerical aspects in your implementation?
Same as with fwdsub, we return -1 in case there is a division of zero, which is not legal.

How did you test your code? What tests did you perform to ensure correctness?
We wrote a script, tri_sylvester_test.c, which dynamically allocates R and C and defines them as matrix structures as inputs for tri_sylvester_solve(). We have checked that the resulting C-matrix is correct according to own solved Sylvester equation, and that it returns -2 for incorrect dimensions as well as -1 for numerical failure with non-legal division.