Exercises sheet 4

by Maximilian Richter and Christian Heppe

Exercise 2

Within the Tutorium we began coding a Gauß elimination algorithm for solving equations of the form $A \cdot \vec{x} = \vec{b}$ with A being $n \times n$ matrix. Since this worked for any given $n \times n$ matrix we continued using the algorithm. It works with three basic loops:

i. Bring the matrix into a upper triangular form.

ii. Bring the matrix into a diagonal form through elimination.

iii. Solve for vector $\vec{x}$.

Since we are using the algortihm in full form we are limited in describing each step without providing subroutines/codes for each excercise step.

1. We implemented the iterative expression for the Gaussian elimination as follows into our algorithm:

   i. $m = \frac{A_{i+1i}}{A_{ii}}$
   $A_{i+1} = A_{i+1} - A_i \cdot m$ and; $i = 1, ..., n$
   where A_i is the i-th row of the matrix. Analogous for $\vec{b}$:
   $b_{i+1} = b_{i+1} - b_i \cdot m$

   ii. $A_i = A_i - \sum_{j=i+1}^{n} k_j \cdot A_j$
   $b_i = b_i - \sum_{j=i+1}^{n} k_j \cdot b_j \; ; i = 1, ..., n; j = i+1, ..., n$
   with $k_j = \frac{A_{ji}}{A_{jj}}$

   iii. $x_i = \frac{b_i}{A_{ii}}$

   Note that we have added several value checks to accomodade for 0's in each step.

2. For a matrix of the given form:
$$\begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{pmatrix}.$$
   we calculate the following
$$c'_i = \begin{cases} \dfrac{c_1}{b_1} & ; \quad i = 1 \\ \dfrac{c_i}{b_i - c'_{i-1} a_i} & ; \quad i = 2, 3, \ldots, n-1 \end{cases}$$
$$d'_i = \begin{cases} \dfrac{d_1}{b_1} & ; \quad i = 1 \\ \dfrac{d_i - d'_{i-1} a_i}{b_i - c'_{i-1} a_i} & ; \quad i = 2, 3, \ldots, n \end{cases}$$

$x_n = d'_n$
$x_i = d'_i - c'_i x_{i+1} \quad ; \; i = n-1, n-2, \ldots, 1$

3. For any given $a_2 \ldots a_n, b_1 \ldots b_n, c_1 \ldots c_{n-1}$ and $y_1 \ldots y_n$ it is necessary to define the matrix as $A = np.array([A_1 \ldots A_n])$ into the code. We didn't do this explicitly because by defining a matrix A the algorithm can solve for $\vec{x}$ with a given $\vec{b}$.
   It should be noted that it is necesary for the check routine to work, to save/copy the initial input values (A as 'AOG' and $\vec{b}$ as 'bOG'). These steps aren't inside our algorithm per se but could easily be implemented into it. For the time being it is necessary to copy the two lines of code under the input for A and $\vec{b}$.

4. As seen in our attached code we implemented a form that would construct us the given matrix as A. The found solution for $\vec{x}$ is given as

$$\vec{x} = (0.5, 0.9, 1.2, 1.4, 1.5, 1.5, 1.4, 1.2, 0.9, 0.5)^T$$

5. With the included check subroutine in our algorithm we found that our result for $\vec{x}$ gives us the exact initial $\vec{y}$, thus we find our algorithm and it's found solution to be satisfactory!