Exercise No. 5

David Bubeck, Pascal Becht, Patrick Nisblè

May 25, 2017

2 - Tridiagonal matrices

1)

Due to the high amount of zeros in the matrix the Gaußian elimination can be simplifyed to

$$\text{for} \quad i = 2...n \begin{cases} a_i = \frac{a_i}{b_{i-1}} \\ b_i = b_i - a_i \cdot c_{i-1} \\ r_i = r_i - a_i \cdot r_{i-1} \end{cases} \tag{1}$$

However, b_1 and c_1 will not be changed.

2)

Now we can use the new found values of b_i and r_i to do a backward substitution. We start the process with the last row, by which the entry of the solution vector is

$$x_n = \frac{r_n}{b_n} \tag{2}$$

For the other entries we obtain with a recursion formula

$$x_i = \frac{r_i - c_i \cdot x_{i+1}}{b_i} \tag{3}$$

3)

To write the code we use the formula above and implement it as a numerical subroutine.

Listing 1: Exercise05.py

```python
import numpy as np


def solve_tridiagonal(a, b, c, r):
    #define length of a as equations we need to calcualte
    numberEquation = len(a)

    #loop for gaussian elimination
    for i in range(1, numberEquation):
        a[i] = a[i] / b[i - 1]
        b[i] = b[i] - a[i] * c[i - 1]
        r[i] = r[i] - a[i] * r[i - 1]

    x = np.zeros(len(b))

    #backward substitution
    x[numberEquation - 1] = r[numberEquation - 1] / b[numberEquation - 1]
    for i in range(numberEquation - 2, -1, -1):
        x[i] = (r[i] - c[i] * x[i + 1]) / b[i]

    return x
```

4)

To test the algorithm we set a matrix with the values to all $a = -1$, all $b = 2$, all $c = -1$ and all $r = 0.1$.

Listing 2: Exercise05.py

```
#define the matrix as give in the sheet
a = np.zeros(9)
b = np.zeros(10)
c = np.zeros(9)
r = np.zeros(10)

for i in range(10):
    b[i] = 2
    r[i] = 0.1
for i in range(9):
    a[i] = -1
    c[i] = -1

aa = np.append([0], [a])
ca = np.append([c], [0])

#copy array values
ac, bc, cc, rc = map(np.array, (aa, b, ca, r))

x = solve_tridiagonal(ac, bc, cc, rc)
print(x)
```

For the vector x we obtain

$$x = \begin{pmatrix} 0.5 \\ 0.9 \\ 1.2 \\ 1.4 \\ 1.5 \\ 1.5 \\ 1.4 \\ 1.2 \\ 0.9 \\ 0.5 \end{pmatrix}$$

0.1 5)

We put the solution back into the original matrix equation to check if the algorithm works correctly. Therefore we use the equations below for a numerical subroutine.

$$a_1 \cdot x_n + b_1 \cdot x_1 + c_1 \cdot x_2 = r_1, \tag{4}$$

$$i = 2, ..., n-1 \quad a_i \cdot x_{i-1} + b_i \cdot x_i + c_i \cdot x_{i+1} = r_i, \tag{5}$$

$$a_n \cdot x_{n-1} + b_n \cdot x_n + c_n \cdot x_1 = r_n \tag{6}$$

Listing 3: Exercise05.py

```
#check if the solution is correct
R = np.zeros(10)
R[0] = b[0] * x[0] + c[0] * x[1]
R[9] = a[-1] * x[-2] + b[9] * x[9]

for i in range(1, 9):
    R[i] = a[i] * x[i - 1] + b[i] * x[i] + c[i] * x[i + 1]
print(R)
```

For the vector r we obtain

$$R = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{pmatrix}$$

We get the same solution we put into our algorithm, $R = r$, therefore is no deviation visible.