Norwegian University of Science
and Technology
Department of Mathematical
Sciences

TMA4135
Matematikk 4D
Fall 2017

Solutions to exercise set 11

1 **a)** We use the FFT implementation in `numpy.fft`. Based on the python code in Listing 1, we find

$$\hat{\vec{f}} = \begin{bmatrix} 12 \\ -3+5\mathrm{i} \\ -6 \\ -3-5\mathrm{i} \end{bmatrix}.$$

Listing 1: Source code for problem 3.

```
import numpy as np
from numpy.fft import fft
import matplotlib.pyplot as plt

f = np.array([0, 2, 3, 7])
N = len(f)
f_hat = fft(f)
print(f_hat)

c = f_hat/N

def DFTinterpolation(x,c):
    q = np.array([0+1j*0]*len(x))
    for n in range(0,N):
        q += c[n]*np.exp(1j*n*x)
    return q

x = np.linspace(0,2*np.pi,1000)
plt.plot(x,DFTinterpolation(x,c).real)

x = np.linspace(0,3/2*np.pi,4)
plt.plot(x,f, linestyle='none', marker="*")
plt.show()
```

b) The coefficients for the trigonometric interpolation polynomial are given by $\vec{c} = \hat{\vec{f}}/N$ (cf. [1, (18) p. 530]). The resulting function

$$q(x) = \sum_{n=0}^{N-1} c_n \mathrm{e}^{\mathrm{i}nx}$$

is plotted in Figure 1 where we can indeed observe the interpolation property.

2 **a)** Extracting the matrix from the linear system of equations yields

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 1 & 4 & -2 \\ 2 & 3 & 8 \end{bmatrix}.$$

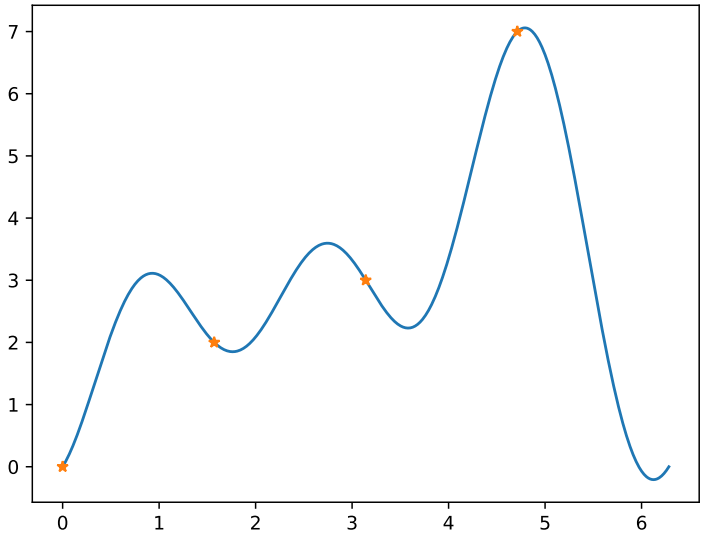

Figure 1: Plot of the real part of the complex trigonometric polynomial.

Since A is strict row diagonally dominant (that is $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ for all i), we know that both Gauss-Seidel and Jacobi iterations will converge to the solution.

b) For the Gauss-Seidel method, the method iterates $\vec{x}^{(m)}$ as follows

$$\begin{aligned} x_1^{(m+1)} &= \frac{1}{5}\left(19 - x_2^{(m)} - 2x_3^{(m)}\right) \\ x_2^{(m+1)} &= \frac{1}{4}\left(-2 - x_1^{(m+1)} + 2x_3^{(m)}\right) \\ x_3^{(m+1)} &= \frac{1}{8}\left(39 - 2x_1^{(m+1)} - 3x_2^{(m+1)}\right). \end{aligned}$$

Starting with $\vec{x}^{(0)} = [1, 1, 1]^\top$ we obtain

$$\vec{x}^{(1)} = \begin{bmatrix} 3.2 \\ -0.8 \\ 4.375 \end{bmatrix}, \quad \vec{x}^{(2)} = \begin{bmatrix} 2.21 \\ 1.135 \\ 3.896875 \end{bmatrix}, \quad \vec{x}^{(3)} = \begin{bmatrix} 2.01425 \\ 0.944875 \\ 4.017109375 \end{bmatrix},$$

$$\vec{x}^{(4)} \approx \begin{bmatrix} 2.00418125 \\ 1.00750938 \\ 3.99613867 \end{bmatrix}.$$

This can be implemented in python as in Listing 1.

c) For the Jacobi method, the method iterates $\vec{x}^{(m)}$ as follows

$$\begin{aligned} x_1^{(m+1)} &= \frac{1}{5}\left(19 - x_2^{(m)} - 2x_3^{(m)}\right) \\ x_2^{(m+1)} &= \frac{1}{4}\left(-2 - x_1^{(m)} + 2x_3^{(m)}\right) \\ x_3^{(m+1)} &= \frac{1}{8}\left(39 - 2x_1^{(m)} - 3x_2^{(m)}\right). \end{aligned}$$

This method can also be written in matrix for as

$$\vec{x}^{(m+1)} = D^{-1}\left(b + (D-A)\vec{x}^{(m)}\right), \quad \text{with} \quad D = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}.$$

Starting with $\vec{x}^{(0)} = [1,1,1]^\top$ we obtain

$$\vec{x}^{(1)} = \begin{bmatrix} 3.2 \\ -0.25 \\ 4.25 \end{bmatrix}, \quad \vec{x}^{(2)} = \begin{bmatrix} 2.15 \\ 0.825 \\ 4.16875 \end{bmatrix}, \quad \vec{x}^{(3)} = \begin{bmatrix} 1.9675 \\ 1.046875 \\ 4.028125 \end{bmatrix},$$
$$\vec{x}^{(4)} = \begin{bmatrix} 1.979375 \\ 1.0221875 \\ 3.990546875 \end{bmatrix}.$$

This can be implemented in python as in Listing 2.

Listing 2: Source code for problem 4.

```
import numpy as np

A = np.array([[5.,1.,2.],
              [1.,4.,-2.],
              [2.,3.,8.]]) # extracted matrix from linear system of equations
b = np.array([[19.],[-2.],[39.]]) # right hand side of linear system of equations

n = len(b)

x = np.array([[1.],[1.],[1.]]) # initial guess
print('For Gauss-Seidel iteration the first 4 iterations are:')
for m in range(0,4):
    for j in range(0,n):
        temp = 0.0 # temporary variable for series summation
        for k in range(0,j):
            temp += A[j,k]*x[k] # Lx^(m+1)
        for k in range(j+1,n):
            temp += A[j,k]*x[k] # Ux^(m)
        x[j] = (b[j]-temp)/A[j,j]
    print(x)

D = np.diag(np.diag(A)) # extract diagonal matrix of A
x = np.array([[1.],[1.],[1.]]) # initial guess
print('For Jacobi iteration the first 4 iterations are:')
for m in range(0,4):
    x = np.linalg.solve(D, b+np.dot(D-A,x))
    print(x)
```

References

[1] E. Kreyszig, *Advanced engineering mathematics*, 10th edition, John Wiley & Sons, 2011.