MA5232 Assignment 1

Feng Qingyang
A0244138H

Contents

1 Introduction

The problem is to find numerical solutions to the following system of partial differential equations defined on $\Omega := (-1,1) \times (-1,1) \times (-1,1)$:

$$\frac{\partial F_1}{\partial x} = \sigma(\frac{1}{6}\sum_{i=1}^{6} F_i - F_1),$$

$$-\frac{\partial F_2}{\partial x} = \sigma(\frac{1}{6}\sum_{i=1}^{6} F_i - F_2),$$

$$\frac{\partial F_3}{\partial y} = \sigma(\frac{1}{6}\sum_{i=1}^{6} F_i - F_3),$$

$$-\frac{\partial F_4}{\partial y} = \sigma(\frac{1}{6}\sum_{i=1}^{6} F_i - F_4),$$

$$\frac{\partial F_5}{\partial z} = \sigma(\frac{1}{6}\sum_{i=1}^{6} F_i - F_5),$$

$$-\frac{\partial F_6}{\partial z} = \sigma(\frac{1}{6}\sum_{i=1}^{6} F_i - F_6),$$

where $\sigma = 0.1, 1, 10, 100$, respectively. The prescribed boundary conditions are:

$$F_1(-1,y,z) = F_b(y,z), F_3(x,-1,z) = F_b(x,z), F_5(x,y,-1) = F_b(x,y),$$

$$F_2(1,y,z) = F_4(x,1,z) = F_6(x,y,1) = 0,$$

where

$$F_b(p,q) = \begin{cases} 1 & |p| \le 0.2 \text{ and } |q| \le 0.2 \\ 0 & \text{otherwise} \end{cases}$$

In general, there does not exist a closed-form solution for such a system, and hence numerical approximated solution is what can be sought for.

In the class a similar 2-dimensional problem has been discussed, and this assignment extend to a 3-dimensional problem, which needs some adjustments to the code for that problem, and demands more computational power. In view of this regard, the method used is Symmetric Gauss-Seidel (SGS) method with Newton iteration, which has a faster convergence. This method has convergent result for all required σ. On the other hand, the source iteration method, and the time evolution method has a slow convergence for large σ (such as $\sigma = 100$), fixed-point iteration method, and SGS method with fixed-point iteration, might diverge for large σ. For the time being, I do not attempt the SGS with synthetic method, which could be a decent choice for this problem.

2 Rationale of SGS with Newton iteration

Given the problem space Ω, the step size (or the number of mesh points) along each axis should be determined first. Then, if using A, B, C, D, E, F to represent the (3-dimensional) discretization matrix for F_1 to F_6, respectively, boundary conditions can be fed into these matrices at corresponding positions.[1] Then SGS method gives six equations:

$$\frac{A_{ijk} - A_{i-1,j,k}}{\Delta x} = \sigma * (\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - A_{ijk}),$$

$$-\frac{B_{i+1,j,k} - B_{ijk}}{\Delta x} = \sigma * (\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E^{+}_{ijk}F_{ijk}) - B_{ijk}),$$

$$\frac{C_{i+1,j,k} - C_{ijk}}{\Delta y} = \sigma * (\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - C_{ijk}),$$

$$-\frac{D_{i+1,j,k} - D_{ijk}}{\Delta y} = \sigma * (\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - D_{ijk}),$$

$$\frac{E_{i+1,j,k} - E_{ijk}}{\Delta z} = \sigma * (\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - E_{ijk}),$$

$$-\frac{F_{i+1,j,k} - F_{ijk}}{\Delta z} = \sigma * (\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - F_{ijk}),$$

where X_{ijk} denotes the (i, j, k) entry of the 3-dimensional matrix X.

However, since the step size in the numerical simulation cannot be arbitrarily small, there would be a residual term in each of the above six equations. In order to reduce residuals, there would be a forward scan (that is, from the first entry not prescribed by boundary conditions to the last such one) and a backward scan to examine the component-wise residual. If such residue is larger than a tolerance value, Newton iteration would be applied to update the corresponding entry of the discretization matrix and hence reduce residuals.

3 Numerical settings

Since the problem space is a direct product of three $(-1, 1)$ intervals and

$$\text{length}((-1, 1)) = 2,$$

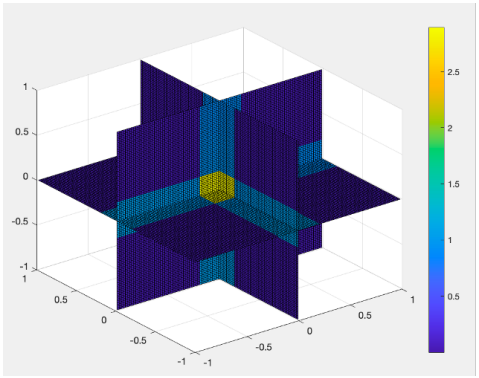
a discretization of 100 evenly-spaced mesh points (hence step size 0.02) inside each interval would be sufficient. Therefore, the problem space Ω is discretized into 100^3 points. Boundary conditions, given by functions of two variables, are evaluated at corresponding grid points. There is a need to compute the norm of matrices, as the method produces matrices with entries representing the residual. The residual of matrices are calculated using Frobenius norm.

For this problem, tolerance of convergence (threshold) is set as 10^{-5} across all σ values examined.

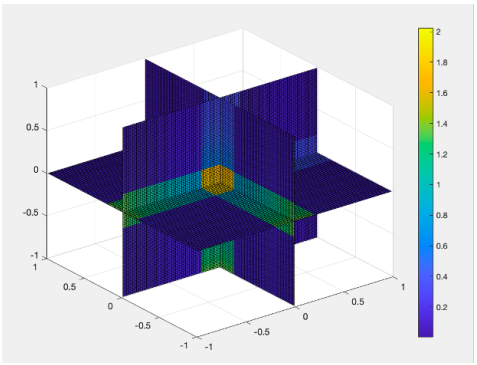
[1] in the code, A_i is used to denote the corresponding matrix for F_i, $1 \leq i \leq 6$.

4 Numerical solutions

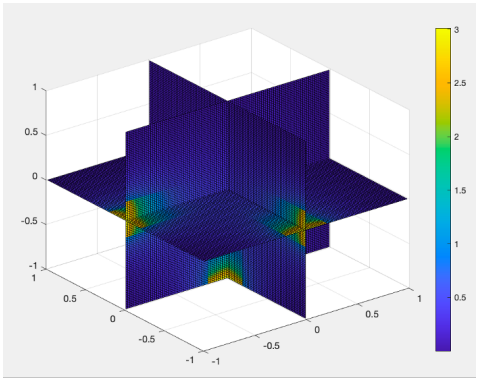
The numerical solutions presented by figures are as follows:



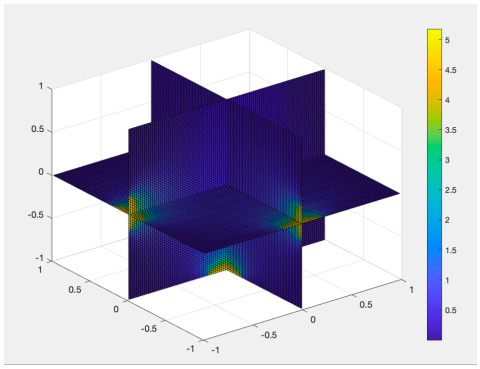
(a) Solution for $\sigma = 0.1$



(b) Solution for $\sigma = 1$



(c) Solution for $\sigma = 10$



(d) Solution for $\sigma = 100$

Figure 1: Plots of numerical solutions for different σ

To interpret these solutions, firstly, notice that at the beginning in the $x = -1$ plane, the value is one in a square centered at $(-1, 0, 0)$ with side length 0.4, and similarly for $y = -1$ and $z = -1$ planes. Remaining positions inside Ω has value zero at the start. Then, as the system evolves, the 'energy' (non-zero values) tend to spread in the space, and as Figure 1 shows, as σ increases, it becomes harder for 'energy' to spread: for example, when $\sigma = 0.1$, the 'energy' spreads to the region around the origin very fast, whereas for $\sigma = 100$, the 'energy' spreads out but decays very fast, so it does not reach the origin.

5 Convergence of the method

Below is a summary of the number of iterations needed to 'converge' (residual less than 10^{-5}) for different σ:

σ	iterations
0.1	2
1	6
10	88
100	1319

Table 1: Number of iterations to converge for different σ

Clearly, the number of iterations needed increases as σ increases. As σ increases, the update tends to be slower and the residual would be larger during the scan process, and thus it takes more time to

converge.

Next have a look of how the method converges as more iterations are processed, as in the following Figure 2:

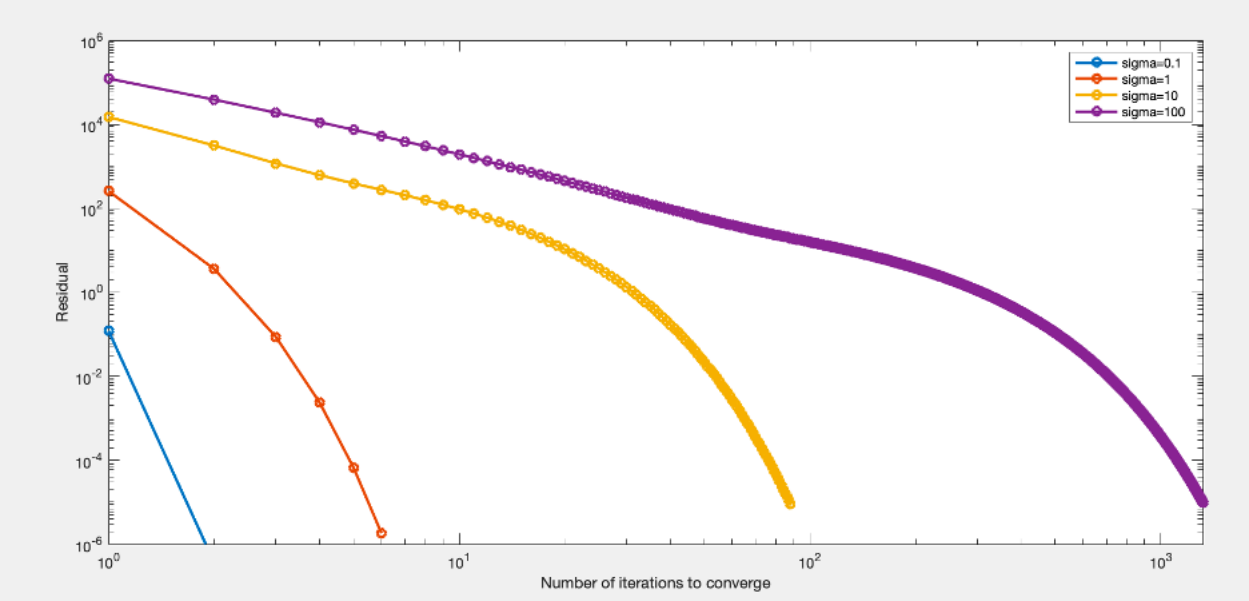


Figure 2: Convergence of SGS with Newton iteration (convergence threshold: 10^{-5})

Since different σ's have varying scale of number of iterations needed to converge, as well as different beginning residual values, for better visualization, both $x-$ and $y-$axis have been changed to log scale, and y starts at 10^{-6}. For each σ, residual decreases significantly during the first several iterations, and then the decrease slows down.

The next section is MATLAB code for this problem. A separate MATLAB code file would be submitted. Also, I have commented the code for plotting the convergence against number of iterations to avoid replacing the numerical solution plot.

6 MATLAB Code

```
function [iter_arr,res_arr,A1,A2,A3,A4,A5,A6] = a1_code(Nx, Ny, Nz, sigma
    ,f1,f2,f3,f4,f5,f6, thres)
%A_i: discretization matrix for F_i; f_i: boundary condition for F_i
dx = 2 / Nx; dy = 2 / Ny; dz = 2 / Nz;
x = linspace(-1+dx/2, 1-dx/2, Nx); %row vector
y = linspace(-1+dy/2, 1-dy/2, Ny);
z = linspace(-1+dz/2, 1-dz/2, Nz);
[X, Y, Z] = meshgrid(x,y,z);

A1 = zeros(Nx+2, Ny+2, Nz+2); A2 = A1; A3 = A1; A4 = A1; A5 = A1; A6 = A1
    ;
n_iter = 0;

% Boundary conditions
[X_xy,Y_xy]=meshgrid(x,y); [X_xz,Z_xz]=meshgrid(x,z);[Y_yz,Z_yz]=meshgrid
    (y,z);
A1b = f1(Y_yz,Z_yz); A2b = f2(Y_yz,Z_yz); A3b = f3(X_xz,Z_xz); A4b = f4(
    X_xz,Z_xz); A5b=f5(X_xy,Y_xy); A6b=f6(X_xy,Y_xy);
A1(1,2:end-1,2:end-1) = A1b'; A2(end,2:end-1,2:end-1) = A2b';
A3(2:end-1,1,2:end-1) = A3b'; A4(2:end-1,end,2:end-1) = A4b';
```

```
A5(2:end-1,2:end-1,1) = A5b'; A6(2:end-1,2:end-1,end) = A6b';

% Residual
resA1 = -(A1(2:Nx+1,2:Ny+1,2:Nz+1) - A1(1:Nx,2:Ny+1,2:Nz+1)) / dx + sigma * ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3(2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A1(2:Nx+1,2:Ny+1,2:Nz+1));
resA2 = (A2(3:Nx+2,2:Ny+1,2:Nz+1) - A2(2:Nx+1,2:Ny+1,2:Nz+1)) / dx + sigma * ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3(2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A2(2:Nx+1,2:Ny+1,2:Nz+1));
resA3 = -(A3(2:Nx+1,2:Ny+1,2:Nz+1) - A3(2:Nx+1,1:Ny,2:Nz+1)) / dy + sigma * ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3(2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A3(2:Nx+1,2:Ny+1,2:Nz+1));
resA4 = (A4(2:Nx+1,3:Ny+2,2:Nz+1) - A4(2:Nx+1,2:Ny+1,2:Nz+1)) / dy + sigma * ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3(2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A4(2:Nx+1,2:Ny+1,2:Nz+1));
resA5 = -(A5(2:Nx+1,2:Ny+1,2:Nz+1) - A5(2:Nx+1,2:Ny+1,1:Nz)) / dz + sigma * ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3(2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A5(2:Nx+1,2:Ny+1,2:Nz+1));
resA6 = (A6(2:Nx+1,2:Ny+1,3:Nz+2) - A6(2:Nx+1,2:Ny+1,2:Nz+1)) / dz + sigma * ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3(2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A6(2:Nx+1,2:Ny+1,2:Nz+1));

res=norm(resA1, 'fro')^2 + norm(resA2, 'fro')^2 + norm(resA3, 'fro')^2 + norm(resA4, 'fro')^2 + norm(resA5, 'fro')^2 + norm(resA6, 'fro')^2;
res_arr = [];
iter_arr=[];

while res > thres && n_iter < 8000
    n_Newton_iter = 0;
    n_nonlin_eqs = 0;

    count=0;
    for i=2:Nx+1
        for j=2:Ny+1
            for k=2:Nz+1
                count=count+1;
                resA1 = -(A1(i,j,k) - A1(i-1,j,k)) / dx + sigma * ...
                    ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k) + A5(i,j,k)+A6(i,j,k))-A1(i,j,k));
                resA2 = (A2(i+1,j,k) - A2(i,j,k)) / dx + sigma * ...
                    ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k) + A5(i,j,k)+A6(i,j,k))-A2(i,j,k));
                resA3 = -(A3(i,j,k) - A3(i,j-1,k)) / dy + sigma * ...
                    ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k) + A5(i,j,k)+A6(i,j,k))-A3(i,j,k));
                resA4 = (A4(i,j+1,k) - A4(i,j,k)) / dy + sigma * ...
                    ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k) + A5(i,j,k)+A6(i,j,k))-A4(i,j,k));
                resA5 = -(A5(i,j,k)-A5(i,j,k-1)) / dz + sigma * ...
                    ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
```

```
            + A5(i,j,k)+A6(i,j,k))-A5(i,j,k));
resA6 = (A6(i,j,k+1)-A6(i,j,k)) / dz + sigma * ...
    ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
        + A5(i,j,k)+A6(i,j,k))-A6(i,j,k));
res = sqrt(resA1^2 + resA2^2 + resA3^2 + resA4^2+ resA5^2
    + resA6^2);

n_nonlin_eqs = n_nonlin_eqs + 1;

while res > thres * 1e-3
    Jac=[1/dx - sigma/6 + sigma, -sigma/6, -sigma/6, -
        sigma/6, -sigma/6, -sigma/6;
        -sigma/6, 1/dx - sigma/6 + sigma, -sigma/6, -
            sigma/6, -sigma/6, -sigma/6;
        -sigma/6, -sigma/6, 1/dy - sigma/6 + sigma, -
            sigma/6, -sigma/6, -sigma/6;
        -sigma/6, -sigma/6, -sigma/6, 1/dy - sigma/6 +
            sigma, -sigma/6, -sigma/6;
        -sigma/6, -sigma/6, -sigma/6, -sigma/6, 1/dz -
            sigma/6 + sigma, -sigma/6;
        -sigma/6, -sigma/6, -sigma/6, -sigma/6, -sigma/6,
             1/dz - sigma/6 + sigma];

    dsol = Jac \ [resA1; resA2; resA3; resA4; resA5;
        resA6];

    A1(i,j,k) = A1(i,j,k) + dsol(1);
    A2(i,j,k) = A2(i,j,k) + dsol(2);
    A3(i,j,k) = A3(i,j,k) + dsol(3);
    A4(i,j,k) = A4(i,j,k) + dsol(4);
    A5(i,j,k) = A5(i,j,k) + dsol(5);
    A6(i,j,k) = A6(i,j,k) + dsol(6);


    resA1 = -(A1(i,j,k) - A1(i-1,j,k)) / dx + sigma * ...
        ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
            j,k) + A5(i,j,k)+A6(i,j,k))-A1(i,j,k));
    resA2 = (A2(i+1,j,k) - A2(i,j,k)) / dx + sigma * ...
        ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
            j,k) + A5(i,j,k)+A6(i,j,k))-A2(i,j,k));
    resA3 = -(A3(i,j,k) - A3(i,j-1,k)) / dy + sigma * ...
        ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
            j,k) + A5(i,j,k)+A6(i,j,k))-A3(i,j,k));
    resA4 = (A4(i,j+1,k) - A4(i,j,k)) / dy + sigma * ...
        ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
            j,k) + A5(i,j,k)+A6(i,j,k))-A4(i,j,k));
    resA5 = -(A5(i,j,k)-A5(i,j,k-1))/dz + sigma * ...
        ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
            j,k) + A5(i,j,k)+A6(i,j,k))-A5(i,j,k));
    resA6 = (A6(i,j,k+1)-A6(i,j,k))/dz + sigma * ...
        ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
            j,k) + A5(i,j,k)+A6(i,j,k))-A6(i,j,k));
    res = sqrt(resA1^2 + resA2^2 + resA3^2 + resA4^2+
        resA5^2 + resA6^2);


```

```
89
90                      n_Newton_iter = n_Newton_iter + 1;
91
92                      if (res > 1e+8)
93                          error("The Newton iteration fails to converge.");
94                      end
95                  end
96              end
97          end
98      end
99
100     for i=Nx+1:-1:2
101         for j=Ny+1:-1:2
102             for k=Nz+1:-1:2
103                 resA1 = -(A1(i,j,k) - A1(i-1,j,k)) / dx + sigma * ...
104                     ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
                            + A5(i,j,k)+A6(i,j,k))-A1(i,j,k));
105                 resA2 = (A2(i+1,j,k) - A2(i,j,k)) / dx + sigma * ...
106                     ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
                            + A5(i,j,k)+A6(i,j,k))-A2(i,j,k));
107                 resA3 = -(A3(i,j,k) - A3(i,j-1,k)) / dy + sigma * ...
108                     ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
                            + A5(i,j,k)+A6(i,j,k))-A3(i,j,k));
109                 resA4 = (A4(i,j+1,k) - A4(i,j,k)) / dy + sigma * ...
110                     ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
                            + A5(i,j,k)+A6(i,j,k))-A4(i,j,k));
111                 resA5 = -(A5(i,j,k)-A5(i,j,k-1))/dz + sigma * ...
112                     ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
                            + A5(i,j,k)+A6(i,j,k))-A5(i,j,k));
113                 resA6 = (A6(i,j,k+1)-A6(i,j,k))/dz + sigma * ...
114                     ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,j,k)
                            + A5(i,j,k)+A6(i,j,k))-A6(i,j,k));
115                 res = sqrt(resA1^2 + resA2^2 + resA3^2 + resA4^2+ resA5^2
                        + resA6^2);
116
117
118                 n_nonlin_eqs = n_nonlin_eqs + 1;
119                 while res > 1e-3 * thres
120                     Jac=[1/dx - sigma/6 + sigma, -sigma/6, -sigma/6, -
                            sigma/6, -sigma/6, -sigma/6;
121                          -sigma/6, 1/dx - sigma/6 + sigma, -sigma/6, -
                                sigma/6, -sigma/6, -sigma/6;
122                          -sigma/6, -sigma/6, 1/dy - sigma/6 + sigma, -
                                sigma/6, -sigma/6, -sigma/6;
123                          -sigma/6, -sigma/6, -sigma/6, 1/dy - sigma/6 +
                                sigma, -sigma/6, -sigma/6;
124                          -sigma/6, -sigma/6, -sigma/6, -sigma/6, 1/dz -
                                sigma/6 + sigma, -sigma/6;
125                          -sigma/6, -sigma/6, -sigma/6, -sigma/6, -sigma
                                /6, 1/dz - sigma/6 + sigma];
126
127                     dsol = Jac \ [resA1; resA2; resA3; resA4; resA5;
                            resA6];
128
129                     A1(i,j,k) = A1(i,j,k) + dsol(1);
130                     A2(i,j,k) = A2(i,j,k) + dsol(2);
```

```
131                     A3(i,j,k) = A3(i,j,k) + dsol(3);
132                     A4(i,j,k) = A4(i,j,k) + dsol(4);
133                     A5(i,j,k) = A5(i,j,k) + dsol(5);
134                     A6(i,j,k) = A6(i,j,k) + dsol(6);
135
136
137                     resA1 = -(A1(i,j,k) - A1(i-1,j,k)) / dx + sigma * ...
138                         ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
                                j,k) + A5(i,j,k)+A6(i,j,k))-A1(i,j,k));
139                     resA2 = (A2(i+1,j,k) - A2(i,j,k)) / dx + sigma * ...
140                         ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
                                j,k) + A5(i,j,k)+A6(i,j,k))-A2(i,j,k));
141                     resA3 = -(A3(i,j,k) - A3(i,j-1,k)) / dy + sigma * ...
142                         ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
                                j,k) + A5(i,j,k)+A6(i,j,k))-A3(i,j,k));
143                     resA4 = (A4(i,j+1,k) - A4(i,j,k)) / dy + sigma * ...
144                         ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
                                j,k) + A5(i,j,k)+A6(i,j,k))-A4(i,j,k));
145                     resA5 = -(A5(i,j,k)-A5(i,j,k-1))/dz + sigma * ...
146                         ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
                                j,k) + A5(i,j,k)+A6(i,j,k))-A5(i,j,k));
147                     resA6 = (A6(i,j,k+1)-A6(i,j,k))/dz + sigma * ...
148                         ((1/6)*(A1(i,j,k) + A2(i,j,k) + A3(i,j,k) + A4(i,
                                j,k) + A5(i,j,k)+A6(i,j,k))-A6(i,j,k));
149
150                     res = sqrt(resA1^2 + resA2^2 + resA3^2 + resA4^2+
                            resA5^2 + resA6^2);
151
152                     n_Newton_iter = n_Newton_iter + 1;
153
154                     if (res > 1e+8)
155                         error("The Newton iteration fails to converge.");
156                     end
157                 end
158             end
159         end
160     end
161
162     resA1 = -(A1(2:Nx+1,2:Ny+1,2:Nz+1) - A1(1:Nx,2:Ny+1,2:Nz+1)) / dx +
            sigma * ...
163         ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3
                (2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:
                Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A1(2:Nx+1,2:Ny+1,2:Nz
                +1));
164     resA2 = (A2(3:Nx+2,2:Ny+1,2:Nz+1) - A2(2:Nx+1,2:Ny+1,2:Nz+1)) / dx +
            sigma * ...
165         ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3
                (2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:
                Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A2(2:Nx+1,2:Ny+1,2:Nz
                +1));
166     resA3 = -(A3(2:Nx+1,2:Ny+1,2:Nz+1) - A3(2:Nx+1,1:Ny,2:Nz+1)) / dy +
            sigma * ...
167         ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3
                (2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:
                Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A3(2:Nx+1,2:Ny+1,2:Nz
                +1));
```

```
168       resA4 = (A4(2:Nx+1,3:Ny+2,2:Nz+1) - A4(2:Nx+1,2:Ny+1,2:Nz+1)) / dy +
              sigma * ...
169           ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3
                  (2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:
                  Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A4(2:Nx+1,2:Ny+1,2:Nz
                  +1));
170       resA5 = -(A5(2:Nx+1,2:Ny+1,2:Nz+1) - A5(2:Nx+1,2:Ny+1,1:Nz)) / dz +
              sigma * ...
171           ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3
                  (2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:
                  Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A5(2:Nx+1,2:Ny+1,2:Nz
                  +1));
172       resA6 = (A6(2:Nx+1,2:Ny+1,3:Nz+2) - A6(2:Nx+1,2:Ny+1,2:Nz+1)) / dz +
              sigma * ...
173           ((1/6)*(A1(2:Nx+1,2:Ny+1,2:Nz+1) + A2(2:Nx+1,2:Ny+1,2:Nz+1) + A3
                  (2:Nx+1,2:Ny+1,2:Nz+1) + A4(2:Nx+1,2:Ny+1,2:Nz+1)+A5(2:Nx+1,2:
                  Ny+1,2:Nz+1)+A6(2:Nx+1,2:Ny+1,2:Nz+1))-A6(2:Nx+1,2:Ny+1,2:Nz
                  +1));
174
175       res=norm(resA1, 'fro')^2 + norm(resA2, 'fro')^2 + norm(resA3, 'fro')
              ^2 + norm(resA4, 'fro')^2 + norm(resA5, 'fro')^2 + norm(resA6, '
              fro')^2;
176
177
178       xslice=0;
179       yslice=0;
180       zslice=0;
181       slice(X,Y,Z,A1(2:end-1,2:end-1,2:end-1)+A2(2:end-1,2:end-1,2:end-1)+
              A3(2:end-1,2:end-1,2:end-1)+A4(2:end-1,2:end-1,2:end-1)+A5(2:end
              -1,2:end-1,2:end-1)+A6(2:end-1,2:end-1,2:end-1),...
182            xslice,yslice,zslice,'nearest');
183       colorbar;
184
185       pause(.1);
186
187       n_iter = n_iter + 1;
188       iter_arr=[iter_arr n_iter];
189       res_arr=[res_arr,res];
190       variableName=num2str(sigma);
191       display=['sigma=',variableName];
192       %plot(iter_arr,res_arr,'o-','LineWidth',2,'DisplayName',display);
193       %xlabel('Number of iterations');
194       %ylabel('Residual');
195       %title('Convergence of the method');
196       %grid on;
197       %legend('show');
198
199       fprintf("Iter %d: Residual: %f, Averge number of Newton iterations: %
              f\n", n_iter, res, n_Newton_iter/n_nonlin_eqs);
200   end
201   end
```

References

Zhenning Cai, *MA5232 lecture slides*

Zhenning Cai, *MA5232 MATLAB codes for lectures*