TMA373 Assignment 6

Edoardo Mangia
edoardom@student.chalmers.se

Bibiana Farinha
bibiana@student.chalmers.se

February 2025

Contents

1 Part 1

The electro-magnetic field in a microwave oven is given by Maxwell's equations:

$$\begin{aligned} \nabla \times \mathbf{E} &= -\mu \mathbf{H}_t \\ \nabla \times \mathbf{H} &= \varepsilon \mathbf{E}_t + \sigma \mathbf{E}. \end{aligned} \tag{1}$$

where $\mathbf{E}$ and $\mathbf{H}$ are the electric and magnetic fields, and the constants ε, σ, μ represent permittivity, electrical conductivity, and permeability, respectively.

Assuming that E is time-harmonic, we get:

$$\mathbf{E} = \mathbf{E}(x, y, z)e^{iwt}, \quad \mathbf{H} = \mathbf{H}(x, y, z)e^{iwt},$$

which we can use to extend 1 further:

$$\begin{aligned}
\nabla \times \mathbf{E} &= -\mu \mathbf{H}_t = -iw\mu \mathbf{H}(x, y, z)e^{iwt} = -iw\mu \mathbf{H}, \\
\nabla \times \mathbf{H} &= \varepsilon \mathbf{E}_t + \sigma \mathbf{E} = iw\varepsilon \mathbf{E} + \sigma \mathbf{E} = iw\tilde{\varepsilon}\mathbf{E},
\end{aligned}$$

where $\tilde{\varepsilon} = \frac{\sigma}{iw} + \varepsilon$ is unknown.

2 Part 2 - redo easy (then we can resubmit)

We can obtain the Helmoltz's equation in 2d, by calculating $\nabla^2 E_3$.

$$\nabla^2 E_3 = \nabla(\nabla E_3) = \nabla H_3 = \nabla(-i\omega\tilde{\varepsilon}E_3) = -i\omega\mu\nabla H_3 = -i\omega\mu \cdot i\omega\tilde{\varepsilon}E_3,$$

since $i = -1/i$, the above simplifies to:

$$\nabla^2 E_3 - \omega^2 \mu \tilde{\varepsilon} E_3 = 0. \tag{2}$$

We shall now discretize 2 with the FEM.

3 Part 3

The first step is to generate a grid. Given the computational domain and frequency of the Helmholtz equation, the implementation below returns a list of nodes, a list of triangles and the material constant $\mu\varepsilon$ for every triangle.

Implementation

```
function [N, T, P] = mygrid(G, w)
epsAir=8.85e-12;
muAir=4*pi*1e-7;
sigmaAir=0;

epsChicken=4.43e-11;
muChicken=4*pi*1e-7;
sigmaChicken=3e-11;

if G==0
    N = [0 0; 1 0; 0 1; 1 1];
    T = [1 2 3 1 0 1; 2 4 3 1 1 0];
    P = [1 1];
elseif G==1
```

```
    N = [0 0; 1 0; 0 1];
    T = [1 2 3 1 1 1];
    P = [1 1];
elseif G==2
    N = [0 0; 1 0; 0 1; 1 1];
    T = [1 2 3 1 0 1; 2 4 3 1 1 0];
    P = muAir * (epsAir + sigmaAir / (1i * w)) * ones(1,2);
elseif G==3
    ... % provided on canvas
end
end
```

4 Part 4

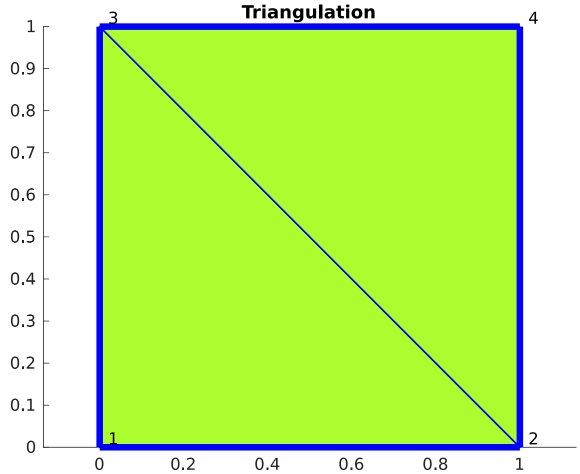
To visualize *mygrid*, we will implement and use the function below *plotmygrid*.

Implementation

```
function plotmygrid(N,T,P)
clf; axis('equal');

for i=1:size(T,1)
    if (abs(P(i)-8.85e-12.*pi*4e-7)<=1e-30)
        patch(N(T(i,1:3),1),N(T(i,1:3),2),[1 1 0]);
    else
        patch(N(T(i,1:3),1),N(T(i,1:3),2),[173 255 47]/256);
    end

    for j=1:3
        line([N(T(i,j),1) N(T(i,mod(j,3)+1),1)], [N(T(i,j),2)
            N(T(i,mod(j,3)+1),2)], ...
            'Color', 'b', 'LineWidth',T(i,j+3)*3+1);
    end
end

m=size(N,1);
if m<100
    for i=1:m
        text(N(i,1)+.02,N(i, 2)+.02,num2str(i));
    end
end
end
```

Testing the implementation for the case where $G = 0$ generated the image below.

Visualization



5 Part 5

To refine the grid generated, we have implemented *mygridrefinement*. Given a list of nodes, a list of triangles and a material constant, the function creates and returns 4 triangles inside each original triangle, creating the necessary nodes as well. The function also saves the information on the boundary and the material constant for each triangle.

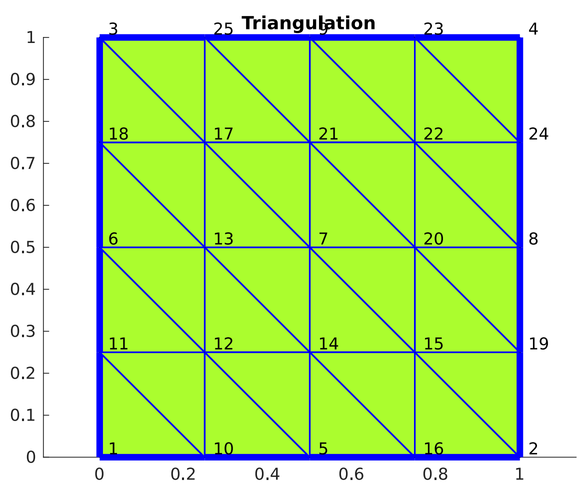
Implementation

```
function [Nr, Tr, Pr] = mygridrefinement(N, T, P);

Nr = N; nn = size(N, 1);
Tr = []; nt = 0; Pr = [];

% Construction of 4 new triangles
for j = 1:size(T, 1)
    i = T(j, 1:3);

    n = [N(i(1), :); N(i(2), :); N(i(3), :); zeros(3, 2)];
    % 3 new nodes
    n(4, :) = (n(1 ,:) + n(2, :))/2;
    n(5, :) = (n(1 ,:) + n(3, :))/2;
    n(6, :) = (n(2 ,:) + n(3, :))/2;

    for k = 4:6
        l = find(Nr(:, 1) == n(k, 1));
        m = find(Nr(l, 2) == n(k, 2));
        if isempty(m)
            nn = nn +1;
```

```
            Nr(nn, :) = n(k, :);
            i(k) = nn ;
        else
            i(k) = l(m);
        end
    end

    % Insert 4 new triangles
    Tr(nt+1, :) = [i(1) i(4) i(5) T(j, 4) 0 T(j, 6)];
    Tr(nt+2, :) = [i(4) i(6) i(5) 0 0 0];
    Tr(nt+3, :) = [i(6) i(4) i(2) 0 T(j, 4) T(j, 5)];
    Tr(nt+4, :) = [i(6) i(3) i(5) T(j, 5) T(j, 6) 0];

    Pr(nt+1:nt+4) = P(j);
    nt = nt + 4;
end
end
```

To test the script, we have used *plotmygrid*, from part 4, which generated the image below.

Visualization



6 Part 6

After we have the desired mesh, we can begin working on the FEM. For that, we need to compute the element mass matrix and the element stiffness matrix.

Given a triangle, the element mass matrix is calculated by:

$$M_e = \frac{J_d}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix},$$

where J_d is the Jacobian determinant of a linear triangle, the area of the triangle:

$$J_d = \frac{1}{2}\left|x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\right|$$

Implementation

```
function Me=elementmassmatrix(t)
x1 = t(1,1); y1 = t(1,2);
x2 = t(2,1); y2 = t(2,2);
x3 = t(3,1); y3 = t(3,2);

Jd = abs( x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2))/2;

Me = Jd/12 * [2 1 1; 1 2 1; 1 1 2];
end
```

Given a triangle, the element stiffness matrix is calculated by:

$$S_e = J_d \cdot D \cdot J^{-1} \cdot J^{-1} \cdot D^T$$

where, the shape function gradients matrix D and the Jacobian matrix J are the following:

$$J = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \quad D = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Implementation

```
function Ke = elementstiffmatrix(t)
x1 = t(1,1); y1 = t(1,2);
x2 = t(2,1); y2 = t(2,2);
x3 = t(3,1); y3 = t(3,2);

b1 = y2 - y3; b2 = y3 - y1; b3 = y1 - y2;
c1 = x3 - x2; c2 = x1 - x3; c3 = x2 - x1;

Area2 = abs(x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2));
Area = Area2 / 2;

Ke = (1/(4*Area)) * [b1^2+c1^2, b1*b2+c1*c2, b1*b3+c1*c3;
                     b2*b1+c2*c1, b2^2+c2^2, b2*b3+c2*c3;
                     b3*b1+c3*c1, b3*b2+c3*c2, b3^2+c3^2];
end
```

7 Part 7

This function is the solver. It takes the boundary condition g, the node/element descriptions, the frequency w, and the material data P. It assembles global matrices $\mathbf{K}$, $\mathbf{M}$, imposes Dirichlet boundary conditions, and solves the linear system

$$(\mathbf{K} - w^2\mathbf{M})\,\mathbf{u} = \mathbf{f}.$$

Here, $\mathbf{f}$ may be zero if the PDE is homogeneous inside, except for boundary effects.

Note that in the implementation $\mathbf{S}$ is the variable equivalent to$\mathbf{K}$.

Implementation

```
function [u, S, M] = FEHelmholtz2D(g, N, T, w, P)

nn = size(N, 1);
nK = size(T, 1);

S = zeros(nn, nn);
M = zeros(nn, nn);
b = zeros(nn, 1);

% compute global matrices
for e = 1:nK
    ie = T(e, 1:3);
    Ne = [N(ie(1),:); N(ie(2), :); N(ie(3),:)];
    Se = elementstiffmatrix(Ne);
    Me = P(e)*elementmassmatrix(Ne);

    for i = 1:3
        for j = 1:3
            S(ie(i), ie(j)) = S(ie(i), ie(j)) + Se(i, j);
            M(ie(i), ie(j)) = M(ie(i), ie(j)) + Me(i, j);
        end
    end
end

A = S - w^2 * M;

% store real boundary indexes
idx_bnd = [];
for e = 1:nK
    ie = T(e, 1:3);
    if T(e, 4) == 1
        idx_bnd = [idx_bnd; ie(1); ie(2)];
    end
    if T(e, 5) == 1
        idx_bnd = [idx_bnd; ie(2); ie(3)];
    end
    if T(e, 6) == 1
        idx_bnd = [idx_bnd; ie(3); ie(1)];
    end
end
```

```
idx_bnd = unique(idx_bnd);

% apply dirichlet BC
for i = idx_bnd'
    b(i) = g(N(i, 1), N(i, 2));
    A(i,:) = 0;
    A(i,i) = 1;
end

u = A \ b;

end
```

8 Part 8

This function takes the final solution vector u and the geometry data (N, T), then creates a 2D visualization (a color map, mesh, or surface plot) of $u_h(x, y)$ over the domain Ω.

Implementation

```
function PlotSolutionHelmholtz(u, N, T)
  figure;
  trisurf(T(:,1:3), N(:,1), N(:,2), real(u), 'FaceColor',
      'interp');
  colorbar;
  xlabel('x'); ylabel('y'); zlabel('u');
  title('FE Solution of the Helmholtz Equation');
  view(3);
end
```

9 Part 9

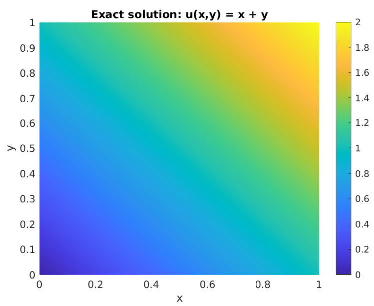
Below is the code tested on the unit square $\Omega = [0, 1] \times [0, 1]$, using $\omega = 0$ and the boundary condition $g(x, y) = x + y$.

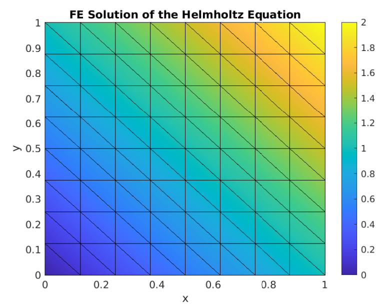
Implementation

```
G = 0;
w = 0;
g = @(x,y) x + y;

[N, T, P] = mygrid(G, w);

numRefinements = 3;
for r = 1:numRefinements
    [N, T, P] = mygridrefinement(N, T, P);
end

figure;
```

```
plotmygrid(N, T, P);
title('Triangulation');

[u, K, M] = FEHelmholtz2D(g, N, T, w, P);
PlotSolutionHelmholtz(u, N, T);
```

Visualization





10 Part 10

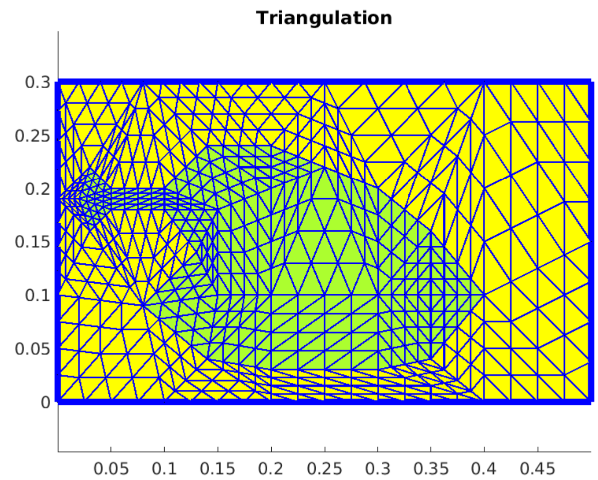
Finally, we ran the code on the "chicken in a microwave" geometry (G=3). We used $\omega = 2\pi \times 2.45\,\text{GHz}$ and assigned

$$g(x, y) = 100 \quad \text{on a portion of the boundary (e.g. } x = 0.5, 0.1 \le y \le 0.2\text{)},$$

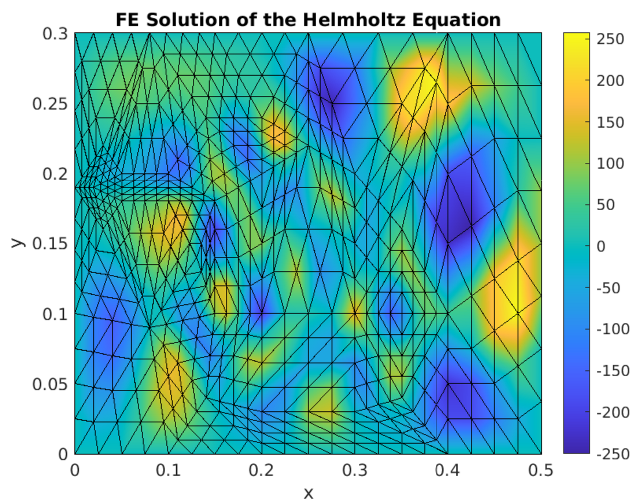
with $g = 0$ elsewhere. Below are also plots representing the real part of the electric field solution.

Implementation

```
w = 2 * (pi*2.45) * 10^9;
g = inline(' 100*( x ==0.5 & 0.1 <= y & 0.2 >= y ) ' , 'x ' , 'y
');
G = 3;

[N, T, P] = mygrid(G, w);

numRefinements = 2;
for r = 1:numRefinements
    [N, T, P] = mygridrefinement(N, T, P);
end

figure;
plotmygrid(N, T, P);
title('Triangulation');

[u, K, M] = FEHelmholtz2Dedo(g, N, T, w, P);
PlotSolutionHelmholtz(u, N, T);
```



Triangulation
0.3
0.25
0.2
0.15
0.1
0.05
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45



FE Solution of the Helmholtz Equation
y
x
0
0.1
0.2
0.3
0.4
0.5
0.05
0.15
0.25
250
200
150
100
50
-50
-100
-150
-200
-250