MATH 228b: HW3

April Novak

March 2, 2017

1

This problem solves the following differential equation:

$$-\nabla^2 u - k^2 u = 0 \tag{1}$$

With boundary conditions:

$$\begin{aligned} \hat{n}\cdot\nabla u =& 0 && \text{on } \Gamma_{wall} \\ \hat{n}\cdot\nabla u =& -iku && \text{on } \Gamma_{out} \\ \hat{n}\cdot\nabla u =& 2ik - iku && \text{on } \Gamma_{in} \end{aligned} \tag{2}$$

where the domain boundary is divided into three sections, where $\Gamma = \Gamma_{wall} \cup \Gamma_{out} \cup \Gamma_{in}$.

1.1 (a)

The weighted residual form is obtained by multiplying the governing equation by a weight function v and integrating over the domain, applying integration by parts when possible:

$$\begin{aligned} -\int_\Omega v\nabla^2 u - \int_\Omega k^2 uv =& 0 \\ \int_\Omega \nabla u\cdot\nabla v - \int_\Gamma \hat{n}v\cdot\nabla u - \int_\Omega k^2 uv =& 0 \end{aligned} \tag{3}$$

Applying the boundary conditions:

$$\int_\Omega \nabla u\cdot\nabla v + \int_{\Gamma_{out}} viku - \int_{\Gamma_{in}} v(2ik - iku) - \int_\Omega k^2 uv = 0 \tag{4}$$

u is expanded in linear continuous polynomials over a series of elements K defined by a triangulation T_h such that:

$$u_h = \{u \in C^0(\Omega) : u|_K \in P_1(K)\forall K \in T_h, \text{for } \hat{n}\cdot\nabla u = 0 \text{ on } \Gamma_{wall}, \hat{n}\cdot\nabla u = -iku \text{ on } \Gamma_{out}, \hat{n}\cdot\nabla u = 2ik - iku \text{ on } \Gamma_{in}\} \tag{5}$$

For a Galerkin approximation, v is chosen from the space V_h, which is the same space as u_h except that V_h satisfy the homogeneous form of the essential boundary conditions (of which there are none in this problem):

$$V_h = \{v \in C^0(\Omega) : v|_K \in P_1(K)\forall K \in T_h\} \tag{6}$$

The Galerkin finite element problem is therefore:

$$\int_\Omega \nabla u_h\cdot\nabla v_h + \int_{\Gamma_{out}} v_h iku_h + \int_{\Gamma_{in}} v_h iku_h - \int_\Omega k^2 u_h v_h = \int_{\Gamma_{in}} v_h 2ik \tag{7}$$

1.2 (b)

u_h is represented as an expansion of basis functions described by the spaces given above:

$$u_h = \sum_{i=1}^{N} a_i \phi_i \tag{8}$$

where N are the number of basis functions over the entire domain. Alternatively, this can be expressed in terms of the solution over each finite element:

$$u_h^e = \sum_{i=1}^{n_{en}} a_i \phi_i \tag{9}$$

where n_{en} are the number of nodes per element and the e superscript indicates that the solution is only over a single element, with piecewise continuity between elements. Inserting these shape functions into the weak form above, where v_h is likewise expanded in the same set of basis functions as u_h, but with expansion coefficients b:

$$\begin{split}\int_\Omega \nabla\left(\sum_{i=1}^{N} a_i\phi_i\right)\cdot\nabla\left(\sum_{j=1}^{N} b_j\phi_j\right) + \int_{\Gamma_{out}}\left(\sum_{j=1}^{N} b_j\phi_j\right) ik\left(\sum_{i=1}^{N} a_i\phi_i\right) + \int_{\Gamma_{in}}\left(\sum_{j=1}^{N} b_j\phi_j\right) ik\left(\sum_{i=1}^{N} a_i\phi_i\right)\\ -\int_\Omega k^2\left(\sum_{i=1}^{N} a_i\phi_i\right)\left(\sum_{j=1}^{N} b_j\phi_j\right) = \int_{\Gamma_{in}}\left(\sum_{j=1}^{N} b_j\phi_j\right)2ik\end{split} \tag{10}$$

A simpler way of representing this weak form is to recognize that u_h and v_h can be represented as:

$$\begin{aligned} u_h =& \mathbf{N}\mathbf{u} = [\phi_1, \phi_2, \cdots, \phi_N]\,[a_1, a_2, \cdots, a_N]^T \\ v_h =& \mathbf{N}\mathbf{v} = [\phi_1, \phi_2, \cdots, \phi_N]\,[b_1, b_2, \cdots, b_N]^T \end{aligned} \tag{11}$$

The gradient of u_h is:

$$\nabla u_h = \mathbf{B}\mathbf{u} = \begin{bmatrix} \frac{\partial\phi_1}{\partial x} & \frac{\partial\phi_2}{\partial x} & \cdots & \frac{\partial\phi_N}{\partial x} \\ \frac{\partial\phi_1}{\partial y} & \frac{\partial\phi_2}{\partial y} & \cdots & \frac{\partial\phi_N}{\partial y} \end{bmatrix}\mathbf{u} \tag{12}$$

The weak form can therefore equivalently be expressed as:

$$\begin{aligned} \int_\Omega (\mathbf{Bu})\cdot(\mathbf{Bv}) + \int_{\Gamma_{out}} (\mathbf{Nv})^T ik(\mathbf{Nu}) + \int_{\Gamma_{in}} (\mathbf{Nv})^T ik(\mathbf{Nu}) - \int_\Omega k^2(\mathbf{Nv})^T(\mathbf{Nu}) = \int_{\Gamma_{in}} (\mathbf{Nv})2ik \\ \int_\Omega (\mathbf{Bv})^T(\mathbf{Bu}) + \int_{\Gamma_{out}} (\mathbf{Nv})^T ik(\mathbf{Nu}) + \int_{\Gamma_{in}} (\mathbf{Nv})^T ik(\mathbf{Nu}) - \int_\Omega k^2(\mathbf{Nv})^T(\mathbf{Nu}) = \int_{\Gamma_{in}} (\mathbf{Nv})^T 2ik \end{aligned} \tag{13}$$

Because $\mathbf{v}^T$ appears in every term, the above could be rearranged such that $\mathbf{v}^T$ multiplies an integrand, which equals zero, which would lead to the conclusion that the integrand itself must equal zero. In other words, $\mathbf{v}^T$ can essentially be canceled from every term.

$$\int_\Omega \mathbf{B}^T(\mathbf{Bu}) + \int_{\Gamma_{out}} \mathbf{N}^T ik(\mathbf{Nu}) + \int_{\Gamma_{in}} \mathbf{N}^T ik(\mathbf{Nu}) - \int_\Omega k^2\mathbf{N}^T(\mathbf{Nu}) = \int_{\Gamma_{in}} \mathbf{N}^T 2ik \tag{14}$$

To obtain a matrix equation in the form of that in the assignment, define:

$$\begin{aligned}
\mathbf{K} &= \int_{\Omega} \mathbf{B}^T \mathbf{B} d\Omega \\
\mathbf{M} &= \int_{\Omega} \mathbf{N}^T \mathbf{N} d\Omega \\
\mathbf{B}_{in} &= \int_{\Gamma_{in}} \mathbf{N}^T \mathbf{N} d\Gamma \\
\mathbf{B}_{out} &= \int_{\Gamma_{out}} \mathbf{N}^T \mathbf{N} d\Gamma \\
\mathbf{b}_{in} &= \int_{\Gamma_{in}} \mathbf{N}^T d\Gamma
\end{aligned} \tag{15}$$

Then, the overall matrix system is:

$$\Big(\mathbf{K} - k^2\mathbf{M} + ik(\mathbf{B}_{in} + \mathbf{B}_{out})\Big)\mathbf{u} = 2ik\mathbf{b}_{in} \tag{16}$$

Refer to the definitions of **N** and **B** to see these definitions in terms of the basis functions.

1.3 (c)

The transmitted intensity, when computed as a function of the finite element solution, is:

$$H(u_h) = \int_{\Gamma_{out}} |\mathbf{N}\mathbf{u}|^2 d\Gamma \tag{17}$$

where $|(.)|$ is the complex absolute value of $(.)$. Because $\mathbf{u}$ may have real and imaginary components, it should be split up as:

$$\mathbf{u} = \mathbf{u}_r + i\mathbf{u}_i \tag{18}$$

where r subscripts indicate the real component and i the imaginary component. Then, the transmitted intensity becomes:

$$\begin{aligned}
H(u_h) &= \int_{\Gamma_{out}} |\mathbf{N}\left(\mathbf{u}_r + i\mathbf{u}_i\right)|^2 d\Gamma \\
&= \int_{\Gamma_{out}} \big(\mathbf{N}(\mathbf{u}_r + i\mathbf{u}_i)\big) \cdot \big(\mathbf{N}(\mathbf{u}_r - i\mathbf{u}_i)\big)\, d\Gamma \\
&= \int_{\Gamma_{out}} \big(\mathbf{N}(\mathbf{u}_r - i\mathbf{u}_i)\big)^T \big(\mathbf{N}(\mathbf{u}_r + i\mathbf{u}_i)\big)\, d\Gamma \\
&= \int_{\Gamma_{out}} (\mathbf{u}_r - i\mathbf{u}_i)^T \mathbf{N}^T \mathbf{N} (\mathbf{u}_r + i\mathbf{u}_i) d\Gamma \\
&= \int_{\Gamma_{out}} \mathbf{u}^H \mathbf{N}^T \mathbf{N} \mathbf{u} d\Gamma \\
&= \mathbf{u}^H \mathbf{B}_{out} \mathbf{u}
\end{aligned} \tag{19}$$

where the definition of $\mathbf{B}_{out}$ from Eq. (27) has been used to show how $H(u)$ can be represented in terms of the finite element solution and $\mathbf{B}_{out}$.

2

This problem solves the problem stated in Eq. (1) for boundary conditions given in Eq. (2) for the rectangular domain given in the assignment.

2.1 (a)

The analytical solution to Eq. (1) with boundary conditions Eq. (2) is only a function of x, since the insulation boundary conditions along the top and bottom edges of the domain are symmetric. Hence, the PDE reduces to an ODE:

$$\frac{d^2u}{dx^2} + k^2u = 0 \tag{20}$$

This common ODE has the following general solution:

$$\begin{aligned} u(x) =& C_1 sin(kx) + C_2 cos(kx) \\ u(x) =& C_1 \frac{1}{2i}\left(e^{ikx} - e^{-ikx}\right) + \frac{C_2}{2}\left(e^{ikx} + e^{-ikx}\right) \end{aligned} \tag{21}$$

Apply the boundary condition on Γ_{in}, where $\hat{n} = [-1, 0]$ and $x = 0$:

$$\begin{aligned} [-1, 0] \cdot (C_1 kcos(kx) - C_2 ksin(kx)) =& 2ik - ik(C_1 sin(kx) + C_2 cos(kx)) \\ C_1 =& -\frac{2ik - ikC_2}{k} \end{aligned} \tag{22}$$

Applying the boundary condition on Γ_{out}, where $\hat{n} = [1, 0]$ and $x = 5$:

$$\begin{aligned} [1, 0] \cdot (C_1 kcos(5k) - C_2 ksin(5k)) =& - ik(C_1 sin(5k) + C_2 cos(5k)) \\ C_1 kcos(5k) - C_2 ksin(5k) =& - ik(C_1 sin(5k) + C_2 cos(5k)) \end{aligned} \tag{23}$$

$$u(x) = -\frac{2 - C_2}{2}\left(e^{ikx} - e^{-ikx}\right) + \frac{C_2}{2}\left(e^{ikx} + e^{-ikx}\right) \tag{24}$$

Applying the boundary condition on Γ_{out}, where $\hat{n} = [1, 0]$ and $x = 5$:

$$\begin{aligned} -\frac{2 - C_2}{2}\left(e^{ikx} + e^{-ikx}\right) + \frac{C_2}{2}\left(e^{ikx} - e^{-ikx}\right) =& \frac{2 - C_2}{2}\left(e^{ikx} - e^{-ikx}\right) - \frac{C_2}{2}\left(e^{ikx} + e^{-ikx}\right) \\ \frac{C_2}{2}\left[\left(e^{ikx} - e^{-ikx} + \left(e^{ikx} + e^{-ikx}\right)\right)\right] =& \frac{2 - C_2}{2}\left[\left(e^{ikx} - e^{-ikx}\right) + \left(e^{ikx} + e^{-ikx}\right)\right] \\ \frac{C_2}{2}\left[e^{ikx} + e^{ikx}\right] =& \frac{2 - C_2}{2}\left[e^{ikx} + e^{ikx}\right] \\ \frac{C_2}{2} =& \frac{2 - C_2}{2} \\ C_2 =& 1 \end{aligned} \tag{25}$$

Then, $C_1 = -i$. Plugging these coefficients into the general solution gives the general solution for these boundaries.

$$\begin{aligned} u(x) =& cos(kx) - isin(kx) \\ u(x) =& e^{-ikx} \end{aligned} \tag{26}$$

2.2 (b)

The function `waveguide_edges` computes arrays of edges for the particular geometry for this question. By first finding all edges, the edges with both x coordinates equal to zero are on Γ_{in}, while edges with both x coordinates equal to 5 are on Γ_{out}, and edges with both y coordinates equal to 0 or 1 are on the Γ_{wall} boundary. The code developed for this section is shown in the Appendix.

2.3 (c)

This section computes the matrices given in Eq. (27) for a given mesh. To facilitate easier use of quadrature rules, all of the integrals in Eq. (27) are transformed to a master element, indicated by superscript e and variables of integration ξ and η, as opposed to x and y in the physical domain. The boundary integrals are slightly more difficult to transform to the master domain. If the boundary corresponds to the bottom of the triangle in the master domain defined with corner points $(\xi = 0, \eta = 0), (\xi = 1, \eta = 0)$, then $\eta = 0$, and the integration is only with respect to ξ (and a normal, 1-D quadrature rule can be used). Likewise, if the boundary corresponds to the left edge of the master domain triangle defined by $(\xi = 0, \eta = 0), (\xi = 0, \eta = 1)$, then $\xi = 0$, and the integration is only with respect to η. For elements with the hypotenuse of the master triangle on a boundary, the connectivity matrix is edited such that the edge along the boundary is the ξ edge (this is performed by simply shuffling the element row in the connectivity matrix by moving every number one position further to the right, and assigning the last number to the first column). Simpson's rule is used for integrating the boundary integrals, while the quadrature rule given in Question 4 is used for the area integrals. The code developed for this section is given in the Appendix. Also included in the Appendix is the code used for swapping location matrix (a.k.a. triangulation) indices.

$$\begin{aligned}
\mathbf{K}^e &= \int_{\Omega^e} (\mathbf{J}^{-1}\mathbf{C})^T(\mathbf{J}^{-1}\mathbf{C})|\mathbf{J}|d\eta d\xi \\
\mathbf{M}^e &= \int_{\Omega^e} \mathbf{N}^T\mathbf{N}|\mathbf{J}|d\eta d\xi \\
\mathbf{B}^e_{in} &= \int_{\Gamma_{in}} \mathbf{N}^T\mathbf{N}d\Gamma \\
\mathbf{B}^e_{out} &= \int_{\Gamma_{out}} \mathbf{N}^T\mathbf{N}d\Gamma \\
\mathbf{b}^e_{in} &= \int_{\Gamma_{in}} \mathbf{N}^T d\Gamma
\end{aligned} \tag{27}$$

where $\mathbf{J}$ and $\mathbf{C}$ are defined as:

$$\begin{aligned}
\mathbf{J} &= \begin{bmatrix} dx/d\xi & dx/d\eta \\ dy/d\xi & dy/d\eta \end{bmatrix} \\
\mathbf{C} &= \begin{bmatrix} \frac{\partial\psi_1}{d\xi} & \frac{\partial\psi_2}{d\xi} & \dots & \frac{\partial\psi_{nen}}{d\xi} \\ \frac{\partial\psi_1}{d\eta} & \frac{\partial\psi_2}{d\eta} & \dots & \frac{\partial\psi_{nen}}{d\eta} \end{bmatrix}
\end{aligned} \tag{28}$$

2.4 (d)

Finally, `femhelmholtz.m` is used to solve the given domain for successively finer and finer meshes to determine the rate of convergence. A short script was developed to show the convergence, and is included in the Appendix. The rate of convergence is estimated as `rate=log2(errors(end-1))-log2(errors(end))`, and is approximately 1.9854. This is to be expected, since the rate of convergence of the finite element method, for sufficiently smooth solutions (which is the case here, since cosines and sines are infinitely differentiable), is only dependent on the polynomial order and an unknown constant. Hence, from the previous assignment, a convergence rate of 2 is obtained for linear elements (and a convergence rate of 3 would be obtained for quadratic elements).

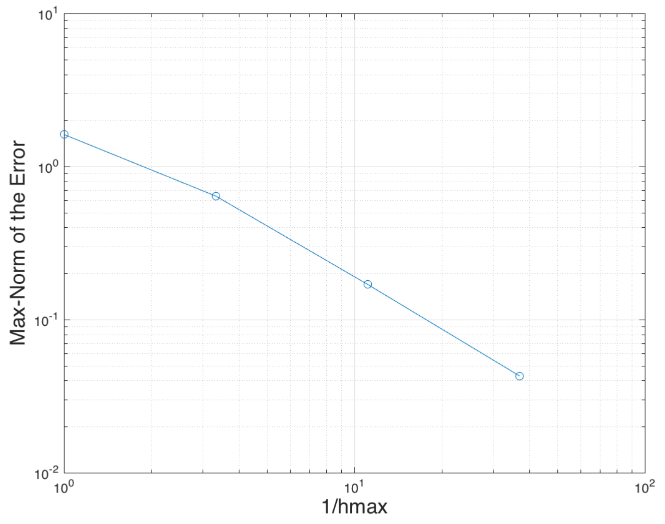

Figure 1. Max norm of the error between the analytic and finite element method solutions.

3

`pmesh.m` (the one developed by the Professor, since I got points off from HW 1 for my `pmesh`) is used to generate a mesh of the domain given in the assignment.

3.1 (a)

Fig. 2 shows the mesh generated for this domain.

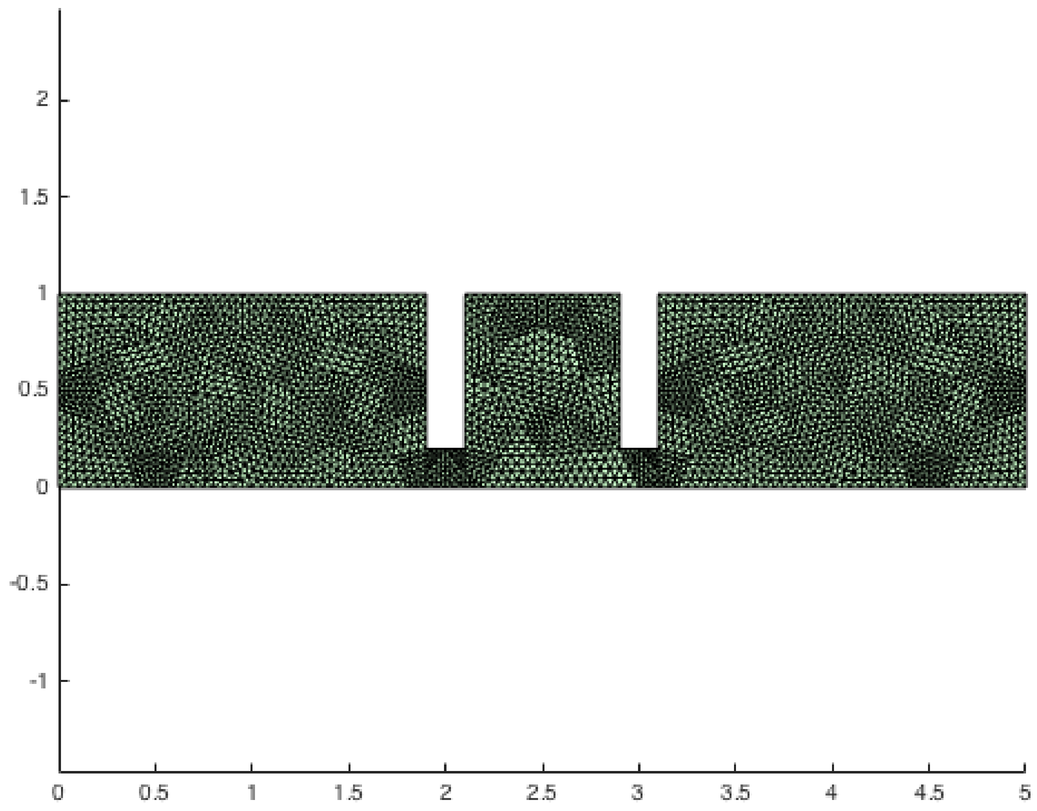

Figure 2. Mesh generated for the provided domain using `pmesh`.

3.2 (b)

In order to investigate wave phenomena, the `femhelmholtz` solver is used to solve Eq. (1) with boundary conditions Eq. (2) on the new domain. To solve using a range of k values, the `femhelmholtz` function is modified so that it can take a k value as input (though this is not the form that is eventually submitted since the format does not fit that requested). It is assumed that all of the boundaries retain the same meaning as in Question 2, and that the boundaries of the slits are also on Γ_{wall}. Fig. 3 shows $log(H(u))$ as a function of k. As can be seen, for $k = 6.36$, there is a minimum in the transmitted intensity near 2π, since there is about two orders or magnitude drop in $H(u)$ from its maximum value over the same range. It is interesting to see that for small changes in k that there is a substantial difference in the transmitted intensity, and this can be attributed to resonance phenomena as waves interact with the geometry of the slits.

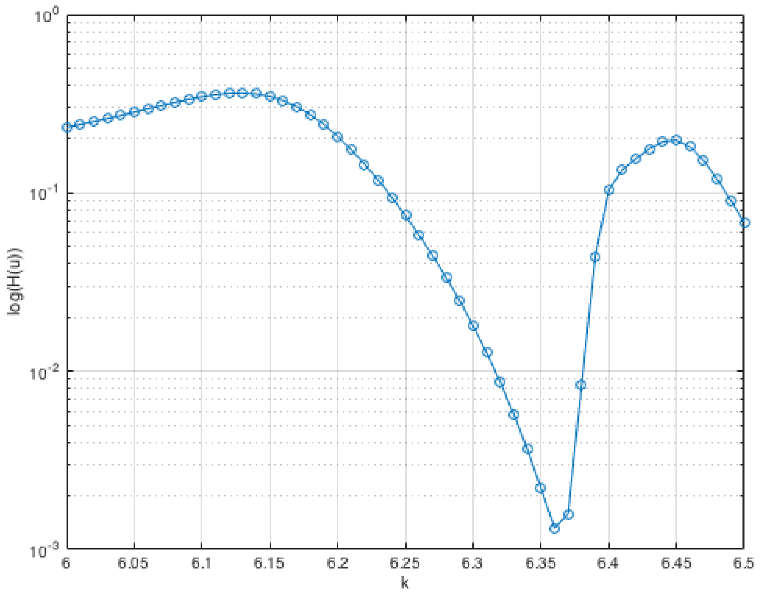

Figure 3. $log(H(u))$ vs. k.

3.3 (c)

The maximum value of $H(u) = 0.3625$ occurs for $k = 6.13$, while the minimum value of $H(u) = 0.0013$ occurs for $k = 6.36$. Fig. 5 shows the real part of the solution obtained for $k = 6.13$, while Fig. **??** shows the solution for $k = 6.36$. As can be seen, the transmitted intensity beyond the first slit for the minimum H is essentially zero, while that intensity is roughly two orders of magnitude greater for the maximum value. All of the code written for this section is included in the Appendix.

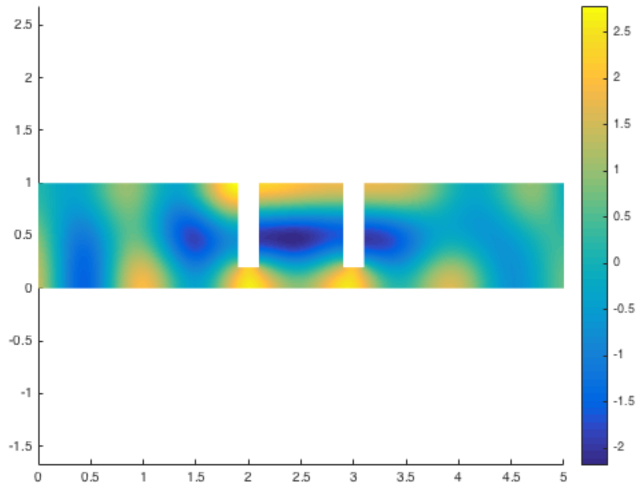

Figure 4. Real part of the solution for $k = 6.13$, or the maximum value in H.

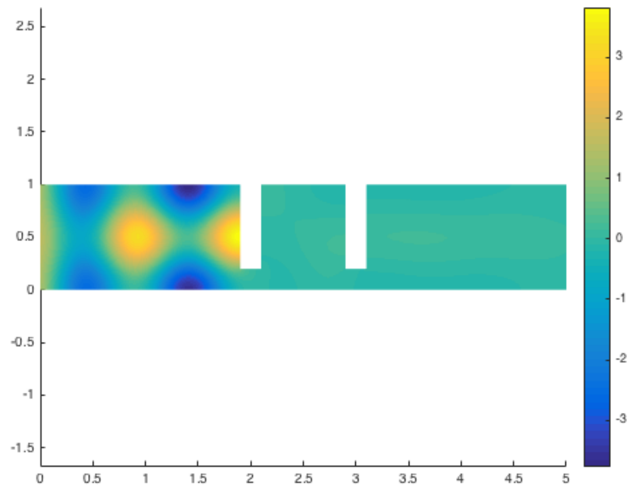

Figure 5. Real part of the solution for $k = 6.36$, or the minimum value in H.

4

This problem extends the `fempoi.m` solver developed in the previous homework for quadratic triangular elements. All triangle are assumed to have straight edges. NOTE: I have assumed a triangulation for nodes numbered 1, 2, 3, 4, 5, 6 in the counterclockwise direction as `t = [1, 2, 3, 4, 5, 6];` for one triangle. The short example given in an email the day before this was due gave a triangulation as `t = [1, 3, 5, 2, 4, 6];`. Using such a triangulation with my code will not work, because I use master domain to physical domain mappings that require a consistent node numbering between the triangulation and the shape functions used in `shapefunctions.m`.

4.1 (a)

A mesh developed for linear basis functions can easily be extended to a mesh for quadratic basis functions by simply adding three additional nodes per triangle, one at the midpoint of each side. The additional points are added to the original mesh by looping over the triangles and adding the midpoints of every side, and the calling `unique` to remove (roughly) half of these points which are shared by two triangles. Then, a loop is created that loops over all of the original triangles and re-computes these midpoints and then searches through the (original + new) list of points to determine the row number of that point. Then, the new triangulation is defined such that the corners of the original triangle fill the 1, 3, and 5 entries of the new triangulation, with the 2, 4, and 6 entries given as the row number of the midpoint along the corresponding edge.

To determine which of the original nodes (in `p`) are on the boundary, `boundary_nodes` is used. Then, to determine which of the new nodes are on the boundary, if two nodes in the same triangle are both on the boundary, then the midpoint node on the edge between those two corner points is in most situations also on the boundary. However, in some cases, when all three corner nodes of a triangle are on the boundary, then at least one of the midpoint nodes is actually not in the boundary. So, a loop over all of the rows and columns in the triangulation is used to determine if a point appears more than once in the triangulation - if so, it is not actually on the boundary. In this way, the boundary nodes, consisting of the original boundary nodes and the new boundary nodes, is created. The code developed for this section is shown in the Appendix.

4.2 (b)

This problem solves the Poisson equation from the previous homework using quadratic Lagrange elements. The equation to be solved is:

$$\begin{gathered}-\nabla^2 u(x,y) = 1\\ u|_{\Gamma_d} = 0, \quad \hat{n}\cdot\nabla u|_{\Gamma_t} = 0\end{gathered} \tag{29}$$

where Γ_d indicates the Dirichlet portion of the boundary and Γ_t the Neumann portion of the boundary, and $\Gamma_d \cup \Gamma_t = \Gamma$. The weighted residual form is obtained by multiplying the above by a weight function v:

$$-\int_\Omega \nabla^2 u(x,y)v(x,y)d\Omega = \int_\Omega v(x,y)d\Omega \tag{30}$$

The weak form is obtained by integrating by parts:

$$\int_\Omega \nabla u(x,y)\cdot\nabla v(x,y)d\Omega - \int_\Gamma \hat{n}\cdot\nabla u(x,y)v(x,y)d\Gamma = \int_\Omega v(x,y)d\Omega \tag{31}$$

Homogeneous Neumann conditions allow the boundary term above to simply be dropped, since $\hat{n}\cdot\nabla u(x,y) = 0$ on Γ_t and the weight function $v(x,y)$ satisfies the homogeneous form of the essential boundary conditions on the essential boundaries such that $v(x,y) = 0$ for all remaining parts of the boundary (since $\Gamma_d \cap \Gamma_t = \emptyset$). After removing this term, the following represents the weak form for the problem, where processing of the global stiffness matrix and global load vector are required to ensure that the solution obtains the specified values on the Dirichlet boundaries.

$$\int_\Omega \nabla u(x,y)\cdot\nabla v(x,y)d\Omega = \int_\Omega v(x,y)d\Omega \tag{32}$$

u is approximated as u_h by expanding it in a series of basis functions ψ:

$$u_h = \sum_{i=1}^{N} a_j\psi_j(x,y) \tag{33}$$

where N is the total number of basis functions, which for Lagrange elements is equivalent to the number of nodes in the domain. Instead of defining basis functions that exist over the entire domain, to improve the sparsity of the matrices involved, expand u_h in n_{en} basis functions over each finite element, where n_{en} are the number of nodes per element.

$$u_h^k = \sum_{i=1}^{n_{en}} a_j\psi_j(x,y) = \mathbf{N}\mathbf{u} \tag{34}$$

The right-hand side can be represented as two vectors, one containing the shape functions, and the other containing the expansion coefficients.

$$\mathbf{N} = \begin{bmatrix}\psi_1(x,y) & \psi_2(x,y) & \cdots & \psi_{n_{en}}\end{bmatrix} \tag{35}$$

$$\mathbf{u} = \begin{bmatrix}a_1 & a_2 & \cdots & a_{n_{en}}\end{bmatrix}^T \tag{36}$$

Likewise, the weight function v can also be expanded in the same manner, but with different expansion coefficients $\mathbf{b}$. The gradient of u_h is defined as:

$$\nabla u_h = \frac{\partial u_h}{\partial x}\hat{x} + \frac{\partial u_h}{\partial y}\hat{y} = \mathbf{B}\mathbf{u} \tag{37}$$

where $\mathbf{B}$ is a deformation matrix that acts on the solution $\mathbf{u}$ to produce the effect of the gradient.

$$\mathbf{B} = \begin{bmatrix}\frac{\partial\psi_1}{dx} & \frac{\partial\psi_2}{dx} & \cdots & \frac{\partial\psi_{nen}}{dx}\\ \frac{\partial\psi_1}{dy} & \frac{\partial\psi_2}{dy} & \cdots & \frac{\partial\psi_{nen}}{dy}\end{bmatrix} \tag{38}$$

Then, noting that the dot product of two vectors can be written as $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}$, the weak form becomes:

$$\begin{aligned} \int_\Omega (\mathbf{Bu})^T (\mathbf{Bv}) d\Omega &= \int_\Omega \mathbf{Nv} d\Omega \\ \int_\Omega \mathbf{B}^T \mathbf{u}^T (\mathbf{Bv}) d\Omega &= \int_\Omega \mathbf{Nv} d\Omega \\ \int_\Omega \mathbf{B}^T \mathbf{uB} d\Omega &= \int_\Omega \mathbf{N} d\Omega \end{aligned} \tag{39}$$

where $\mathbf{v}$ has essentially been cancelled from every term (the above could be rearranged such that $\mathbf{v}$ acts on an integrand, where the entire term equals zero such that the integrand must therefore equal zero). The above is essentially equivalent to the 1-D form, where $\mathbf{B}$ would reduce to $d\psi_i/dx$ and $\mathbf{N}$ to ψ_i. The elemental stiffness matrix and elemental load vector are:

$$\mathbf{A}^k = \int_{\Omega_k} \mathbf{B}^T \mathbf{B} d\Omega \tag{40}$$

$$\mathbf{F}^k = \int_{\Omega_k} \mathbf{N} d\Omega \tag{41}$$

In order to apply over an individual element, the shape functions that appear in $\mathbf{B}$ and $\mathbf{N}$ must be the shape functions over that particular element. This can be achieved either by determining the shape functions over each element in the physical domain (in which case the shape functions are different for every element) or by using isoparametric elements that are mapped from a master domain, where the integrals above are always the same, and then modify the above terms with a Jacobian of the transformation. Therefore, a key difference between `fempoi2` and `fempoi` is the selection of these boundary conditions. To determine these shape functions over the master element, a quadratic shape function has the form $\psi(x) = a+bx+cy+dx^2+ey^2+fxy$. If these shape functions are determined individually for each element, they would be determined by solving the following system for the six different right-hand sides shown below to give the coefficients a, b, c, d, e, f for each of the six shape functions (assuming quadratic triangular elements for this problem):

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & y_1^2 & x_1y_1 \\ 1 & x_2 & y_2 & x_2^2 & y_2^2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3^2 & y_3^2 & x_3y_3 \\ 1 & x_4 & y_4 & x_4^2 & y_4^2 & x_4y_4 \\ 1 & x_5 & y_5 & x_5^2 & y_5^2 & x_5y_5 \\ 1 & x_6 & y_6 & x_6^2 & y_6^2 & x_6y_6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{42}$$

where x_i, y_i is the i coordinate of the triangle, for $i = 1, 2, 3, 4, 5, 6$. Performing this for every triangle individually is tedious, and likewise determining the appropriate bounds of integration for each triangle is problematic and difficult when only the coordinate points are known. To overcome this, a mapping is performed from a master triangle with corners at $(0, 0), (1, 0), (0, 1)$ and midpoints at $(0.5, 0), (0.5, 0.5), (0, 0.5)$ in a coordinate system defined in terms of variables ξ and η. The shape functions in this master domain will therefore be the same over each element in the physical domain, and the only thing that differs for each element is the Jacobian of the transformation. With $(x_1 = 0, y_1 = 0), (x_2 = 0.5, y_2 = 0), (x_3 = 1, y_3 = 0), (x_4 = 0.5, y_4 = 0.5), (x_5 = 0, y_5 = 1), (x_6 = 0, y_6 = 0.5)$, the shape functions in the master domain are:

$$\begin{aligned} \psi_1(\xi,\eta) =& 1 - 3\xi - 3\eta + 2\xi^2 + 2\eta^2 + 4\xi\eta \\ \psi_2(\xi,\eta) =& 4\xi - 4\xi^2 - 4\xi\eta \\ \psi_3(\xi,\eta) =& -\xi + 2\xi^2 \\ \psi_4(\xi,\eta) =& 4\xi\eta \\ \psi_5(\xi,\eta) =& -\eta + 2\eta^2 \\ \psi_6(\xi,\eta) =& 4\eta - 4\eta^2 - 4\xi\eta \end{aligned} \tag{43}$$

The mapping from the master to physical domain is performed using an isoparametric mapping that relates the physical coordinates (X_i, Y_i) to the master domain coordinates (ξ, η)

$$\begin{aligned} x(\xi,\eta) =& \sum_{i=1}^{n_{en}} X_i \psi_i(\xi,\eta) \\ y(\xi,\eta) =& \sum_{i=1}^{n_{en}} Y_i \psi_i(\xi,\eta) \end{aligned} \tag{44}$$

To transform the integrals appearing in Eq. (47) and (48) to integrals over the master domain requires the use of the derivatives with respect to ξ and η, rather than with respect to x and y. The domain of integration changes according to:

$$\begin{aligned} d\vec{x} =& \mathbf{D} d\vec{\xi} \\ \begin{bmatrix} dx \\ dy \end{bmatrix} =& \begin{bmatrix} dx/d\xi & dx/d\eta \\ dy/d\xi & dy/d\eta \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \end{aligned} \tag{45}$$

where $\mathbf{D}$ is the transformation matrix that describes how the coordinate transformation is performed. The derivatives appearing in $\mathbf{D}$ are computed based on the definitions of the isoparametric mapping:

$$\begin{aligned} \frac{dx(\xi,\eta)}{d\xi} =& \sum_{i=1}^{n_{en}} X_i \frac{d\psi_i(\xi,\eta)}{d\xi} \\ \frac{dx(\xi,\eta)}{d\eta} =& \sum_{i=1}^{n_{en}} X_i \frac{d\psi_i(\xi,\eta)}{d\eta} \\ \frac{dy(\xi,\eta)}{d\xi} =& \sum_{i=1}^{n_{en}} Y_i \frac{d\psi_i(\xi,\eta)}{d\xi} \\ \frac{dy(\xi,\eta)}{d\eta} =& \sum_{i=1}^{n_{en}} Y_i \frac{d\psi_i(\xi,\eta)}{d\eta} \end{aligned} \tag{46}$$

With these definitions, the local stiffness matrix and local load vector transform to integrals over ξ, η:

$$\mathbf{A}^k = \int_0^1 \int_0^{1-\xi} (\mathbf{D}^{-1}\mathbf{C})^T (\mathbf{D}^{-1}\mathbf{C}) |\mathbf{D}| d\eta d\xi \tag{47}$$

$$\mathbf{F}^k = \int_0^1 \int_0^{1-\xi} \mathbf{N} |\mathbf{D}| d\eta d\xi \tag{48}$$

where $\mathbf{C}$ is the same as $\mathbf{B}$, except that derivatives are taken with respect to ξ and η instead of x and y:

$$\mathbf{C} = \begin{bmatrix} \frac{\partial \psi_1}{d\xi} & \frac{\partial \psi_2}{d\xi} & \dots & \frac{\partial \psi_{n_en}}{d\xi} \\ \frac{\partial \psi_1}{d\eta} & \frac{\partial \psi_2}{d\eta} & \dots & \frac{\partial \psi_{n_en}}{d\eta} \end{bmatrix} \tag{49}$$

Note that the integrals in Eq. (47) and (48) are no longer over the domain of an element in the physical domain - they are over a triangle in the master domain. To allow easy calculation of these integrals without the need to use symbolic integration or the need to hard-code in the integrals given that all of the shape functions are linear in x and y (to permit faster extension to higher-order elements), quadrature rules are used to numerically evaluate the integrals above. A second motivation to using the transformation to the master domain is the ability to use the same quadrature rule over every element. Quadrature rules for the standard triangle defined with corners at $(0,0),(1,0),(0,1)$ are attempting to determine the weights w_i and quadrature points ξ_i, η_i so that the following summation is as accurate as possible for a polynomial of some given order:

$$\int_0^1 \int_0^{1-\xi} f(\xi,\eta) d\eta d\xi \approx \sum_{i=1}^{n_{qp}} w_i f(\xi_i, \eta_i) \tag{50}$$

For $n_{qp} = 3$, the following three-point rule exactly integrates quadratic integrands over the master domain with corners at $(\xi_1 = 0, \eta_1 = 0), (\xi_2 = 1, \eta_2 = 0), (\xi_3 = 0, \eta_3 = 1)$.

$$
\begin{aligned}
w &= \begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix} \\
(\xi_i, \eta_i) &= \begin{bmatrix} \sum_{j=1}^{3} \left(\frac{\xi_j}{6} + \frac{\xi_1}{2}\right) & \sum_{j=1}^{3} \left(\frac{\eta_j}{6} + \frac{\eta_1}{2}\right) \\ \sum_{j=1}^{3} \left(\frac{\xi_j}{6} + \frac{\xi_2}{2}\right) & \sum_{j=1}^{3} \left(\frac{\eta_j}{6} + \frac{\eta_2}{2}\right) \\ \sum_{j=1}^{3} \left(\frac{\xi_j}{6} + \frac{\xi_3}{2}\right) & \sum_{j=1}^{3} \left(\frac{\eta_j}{6} + \frac{\eta_3}{2}\right) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{6}(\xi_1 + \xi_2 + \xi_3) & \frac{1}{6}(\eta_1 + \eta_2 + \eta_3) \\ \frac{1}{6}(\xi_1 + \xi_2 + \xi_3) + \frac{3}{2} & \frac{1}{6}(\eta_1 + \eta_2 + \eta_3) \\ \frac{1}{6}(\xi_1 + \xi_2 + \xi_3) & \frac{1}{6}(\eta_1 + \eta_2 + \eta_3) + \frac{3}{2} \end{bmatrix} \\
&= \begin{bmatrix} 1/6 & 1/6 \\ 1/6 + 2/3 & 1/6 \\ 1/6 & 1/6 + 2/3 \end{bmatrix}
\end{aligned} \tag{51}
$$

Then, the quadrature-integrated integral is multiplied by the area of the triangle, which is 1/2 for the master triangle. The elemental stiffness matrix and elemental load vector are computed for each individual element. They are then assembled in the global matrix according to the connectivity matrix, which is conveniently provided as the triangulation returned by the `delaunay` function. For example, with the following triangulation for a domain where each element has four nodes, the connectivity matrix, also called the location matrix (LM), would be:

$$
\mathbf{LM} = \begin{bmatrix} 1 & 2 & 5 & 4 \\ 2 & 3 & 6 & 5 \\ 4 & 5 & 8 & 7 \\ 5 & 6 & 9 & 8 \end{bmatrix} \tag{52}
$$

where the local nodes are numbered in a counterclockwise manner beginning from the bottom left node. Then, for example, the second row in the global stiffness matrix would be assembled as:

$$
\mathbf{K}(2,:) = \begin{bmatrix} k_{2,1}^{e=1}, & k_{2,2}^{e=1} + k_{1,1}^{e=2}, & k_{1,2}^{e=2}, & k_{2,4}^{e=1}, & k_{1,4}^{e=2} + k_{2,3}^{e=1}, & k_{1,3}^{e=2}, & 0, & 0, & 0 \end{bmatrix} \tag{53}
$$

Then, similar to the discussion in Problem 2, instead of solving $\mathbf{Au} = \mathbf{f}$ using the full matrices above, the following system must be solved:

$$
A_{uu} u_u = f_u - A_{uk} x_k \tag{54}
$$

Instead of the above approach of solving the matrix system, where post-processing must be performed to apply Dirichlet conditions, the Dirichlet conditions can also be applied directly into the matrices $\mathbf{A}$ and $\mathbf{f}$ - this is the approach used in this assignment. Fig. 6 shows the solution for the same domain as in homework 2. The code used for this section is given in the Appendix.

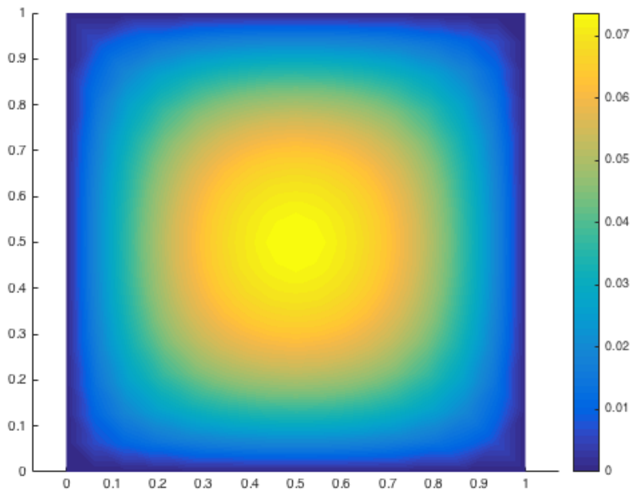

Figure 6. Solution of the Poisson problem over the same domain as in homework 2.

4.3 (c)

Finally, a convergence study is performed over the domain shown above for a maximum refinement level of 4, where the errors are compared against the finest mesh over the node points that are common to the coarsest and finest meshes. Fig. 7 shows on a log-log plot the max-norm of the error as a function of $1/h_{max}$. By computing the rate of convergence as `rate=log2(errors(end-1))-log2(errors(end))`, the convergence rate is 3.24. The expected convergence rate is 3 due to the use of quadratic elements, and likely if the error were compared against a finer mesh with higher than four refinements, this would be observed, but because we are calculating errors relative to a numerical solution, which itself has errors, it is not that surprising that the theoretical convergence rate is not observed. The script used to measure the convergence rate is given in the Appendix.

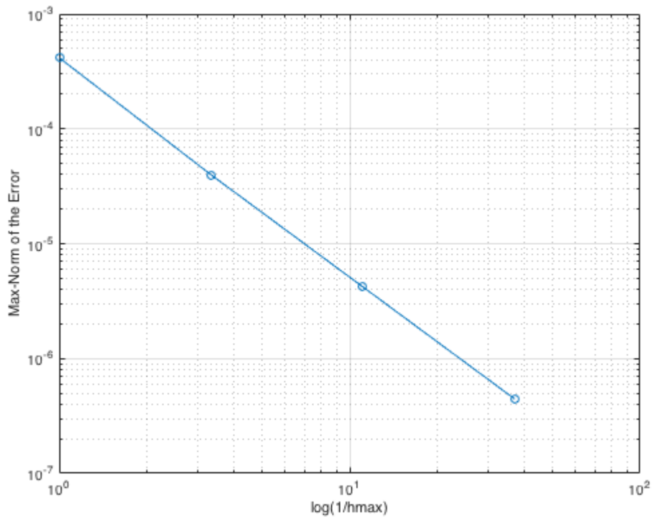

Figure 7. Max norm of the error compared with the finest mesh for a maximum refinement level of 4.

5 Appendix

5.1 Question 2

5.1.1 Part (b) - waveguide_edges.m

```
function [ein, eout, ewall] = waveguide_edges(p, t)
tol = 1e-12;

% find all mesh edges - from boundary_nodes.m

edges = [t(:,[1,2]);
         t(:,[2,3]);
         t(:,[3,1])];
edges = sort(edges, 2);
[C, ia, ic] = unique(edges, 'rows');

ein = [];
eout = [];
ewall = [];

for i = 1:length(C)
    flag = 0;

    % find vertical edges at x = 0
    if p(C(i, 1), 1) == 0 || p(C(i, 2), 1) == 0
        if abs(p(C(i, 1), 1) - p(C(i, 2), 1)) >= tol
        else
            ein = [ein; C(i, :)];
            flag = 1;
        end
    end

    % find vertical edges at x = 5
    if p(C(i, 1), 1) == 5 || p(C(i, 2), 1) == 5
        if abs(p(C(i, 1), 1) - p(C(i, 2), 1)) >= tol
        else
            eout = [eout; C(i, :)];
            flag = 1;
        end
    end

    % find horizontal edges with y = 0
    if p(C(i, 1), 2) == 0 || p(C(i, 2), 2) == 0
        if abs(p(C(i, 1), 2) - p(C(i, 2), 2)) >= tol
        else
            ewall = [ewall; C(i, :)];
            flag = 1;
        end
    end

    % find horizontal edges with y = 1
    if p(C(i, 1), 2) == 1 || p(C(i, 2), 2) == 1
        if abs(p(C(i, 1), 2) - p(C(i, 2), 2)) >= tol
```

```
          else
              ewall = [ewall; C(i, :)];
              flag = 1;
          end
      end
end

% for i = 1:length(ein(:,1))
%     scatter(p(ein(i, 1), 1), p(ein(i, 1), 2), 'ro')
%     hold on
%     scatter(p(ein(i, 2), 1), p(ein(i, 2), 2), 'ro')
%     hold on
% end

% for i = 1:length(eout(:,1))
%     scatter(p(eout(i, 1), 1), p(eout(i, 1), 2), 'bo')
%     hold on
%     scatter(p(eout(i, 2), 1), p(eout(i, 2), 2), 'bo')
%     hold on
% end

% for i = 1:length(ewall(:,1))
%     scatter(p(ewall(i, 1), 1), p(ewall(i, 1), 2), 'go')
%     hold on
%     scatter(p(ewall(i, 2), 1), p(ewall(i, 2), 2), 'go')
%     hold on
% end
```

5.1.2 Part (c) - femhelmholtz.m

```
function [K, M, Bin, Bout, bin] = femhelmholtz(p, t, ein, eout)

num_elem = length(t(:,1));          % number of elements
num_nodes_per_elem = 3;             % linear triangular elements
num_nodes = length(p(:,1));         % total number of nodes

% form the permutation matrix for assembling the global matrices
[perm] = permutation(num_nodes_per_elem);

% 1-D quadrature rule (Simpson's rule)
wt1 = [1/6, 4/6, 1/6];
qp1 = [0, 0.5, 1];

% two-point rule - already multiplied by the area
wt = 0.5 .* [1/3; 1/3; 1/3];
qp = [1/6,1/6; 2/3,1/6; 1/6,2/3];

K = sparse(num_nodes, num_nodes); M = sparse(num_nodes, num_nodes);
Bin = sparse(num_nodes, num_nodes); Bout = sparse(num_nodes, num_nodes); F =
    ↪ zeros(num_nodes, 1);

% change the t for weird elements
[t] = changeLM(num_elem, t, ein, eout);
```

```
for elem = 1:num_elem
    k = 0; m = 0; bout = 0; bin = 0; bright = 0;

    for ll = 1:length(wt)
        [N, dN_dxe, dN_deta, x_xe_eta, y_xe_eta, dx_dxe, dx_deta, dy_dxe,
           ↪ dy_deta, B] = shapefunctions(qp(ll, 1), qp(ll, 2),
           ↪ num_nodes_per_elem, p, t, elem);
        F_mat = transpose([dx_dxe, dx_deta; dy_dxe, dy_deta]);
        J = det(F_mat);
        D = inv(F_mat) * B;

        k = k + wt(ll) * transpose(D) * (D) * J;
        m = m + wt(ll) * N * transpose(N) * J;
    end

    % find the edges of the current element
    edges = [t(elem,[1,2]); t(elem,[2,3]); t(elem,[3,1])];
    edges = sort(edges, 2);

    xe_edge = [t(elem, 1), t(elem, 2)];
    eta_edge = [t(elem, 3), t(elem, 1)];
    xe_edge = sort(xe_edge, 2);
    eta_edge = sort(eta_edge, 2);

    in_flag = 0; out_flag = 0;
    for edge = 1:length(edges(:,1))

        for in = 1:length(ein(:, 1))      % is it on the ein boundary?
            if edges(edge, 1) == ein(in, 1) && edges(edge, 2) == ein(in, 2)
                in_flag = 1;
                for l = 1:length(wt1)
                    if edges(edge, 1) == xe_edge(1) && edges(edge, 2) ==
                       ↪ xe_edge(2)
                        xe = qp1(l); eta = 0;
                    else
                        xe = 0; eta = qp1(l);
                    end
                    [N, dN_dxe, dN_deta, x_xe_eta, y_xe_eta, dx_dxe, dx_deta,
                       ↪ dy_dxe, dy_deta, B] = shapefunctions(xe, eta,
                       ↪ num_nodes_per_elem, p, t, elem);
                    dx = p(edges(edge, 1), 1) - p(edges(edge, 2), 1);
                    dy = p(edges(edge, 1), 2) - p(edges(edge, 2), 2);
                    J = sqrt(dx^2 + dy^2);
                    bin = bin + wt1(l) * N * transpose(N) * J;
                    bright = bright + wt1(l) * N * J;
                end
            end
        end

        % is it on the eout boundary?
        for in = 1:length(eout(:, 1))
            if edges(edge, 1) == eout(in, 1) && edges(edge, 2) == eout(in, 2)
                out_flag = 1;
```

```
                    for l = 1:length(wt1)
                        if edges(edge, 1) == xe_edge(1) && edges(edge, 2) ==
                            ↪ xe_edge(2)
                            xe = qp1(l); eta = 0;
                        else
                            xe = 0; eta = qp1(l);
                        end
                        [N, dN_dxe, dN_deta, x_xe_eta, y_xe_eta, dx_dxe, dx_deta,
                            ↪ dy_dxe, dy_deta, B] = shapefunctions(xe, eta,
                            ↪ num_nodes_per_elem, p, t, elem);
                        dx = p(edges(edge, 1), 1) - p(edges(edge, 2), 1);
                        dy = p(edges(edge, 1), 2) - p(edges(edge, 2), 2);
                        J = sqrt(dx^2 + dy^2);
                        bout = bout + wt1(l) * N * transpose(N) * J;
                    end
                end
            end

            % is it on the Wall boundary? - do nothing, homogeneous Neumann
        end

        % place the elemental k matrix into the global K matrix
        for mm = 1:length(perm(:,1))
            i = perm(mm,1);
            j = perm(mm,2);
            K(t(elem, i), t(elem, j)) = K(t(elem, i), t(elem, j)) + k(i,j);
            M(t(elem, i), t(elem, j)) = M(t(elem, i), t(elem, j)) + m(i,j);
            if (in_flag)
                Bin(t(elem, i), t(elem, j)) = Bin(t(elem, i), t(elem, j)) + bin(i,j
                    ↪ );
            end
            if (out_flag)
                Bout(t(elem, i), t(elem, j)) = Bout(t(elem, i), t(elem, j)) + bout(
                    ↪ i,j);
            end
        end

        % place the elemental f vector into the global F vector
        for i = 1:num_nodes_per_elem
            if (in_flag)
                F(t(elem, i)) = F((t(elem, i))) + bright(i);
            end
        end
end

bin = F;
end
```

5.1.3 Part (c) - `changeLM.m`

```
function [LM] = changeLM(num_elem, LM, In, Out)
% change the LM for weird elements from 1 - 2 - 3 (2 - 3 is on boundary)
% to 2 - 3 - 1 (edge 2-3 is now the xe edge)
```

```
for elem = 1:num_elem
    % find the edges of the current element
    edges = [LM(elem,[1,2]); LM(elem,[2,3]); LM(elem,[3,1])];
    edges = sort(edges, 2);
    xe_edge = [LM(elem, 1), LM(elem, 2)];
    eta_edge = [LM(elem, 3), LM(elem, 1)];
    xe_edge = sort(xe_edge, 2);
    eta_edge = sort(eta_edge, 2);

    for edge = 1:length(edges(:,1))
        % is it on the In boundary?
        for in = 1:length(In(:, 1))
            if edges(edge, 1) == In(in, 1) && edges(edge, 2) == In(in, 2)
                if edges(edge, 1) == xe_edge(1) && edges(edge, 2) == xe_edge
                   ↪ (2)
                elseif edges(edge, 1) == eta_edge(1) && edges(edge, 2) ==
                   ↪ eta_edge(2)
                else
                    %sprintf('Changing LM for element %i', elem)
                    LM(elem, :) = [LM(elem, 2), LM(elem, 3), LM(elem, 1)];
                end
            end
        end

        % is it on the Out boundary?
        for in = 1:length(Out(:, 1))
            if edges(edge, 1) == Out(in, 1) && edges(edge, 2) == Out(in, 2)
                if edges(edge, 1) == xe_edge(1) && edges(edge, 2) == xe_edge
                   ↪ (2)
                elseif edges(edge, 1) == eta_edge(1) && edges(edge, 2) ==
                   ↪ eta_edge(2)
                else
                    %sprintf('Changing LM for element %i', elem)
                    LM(elem, :) = [LM(elem, 2), LM(elem, 3), LM(elem, 1)];
                end
            end
        end
    end
end

end
```

5.1.4 Part (d) - `convergence_helmholtz.m`

```
% script for convergence study of femhelmholtz
clear all

wave = 6;
imag = 1i;
hmax = 0.3;
nrefmax = 4;
errors = zeros(1, nrefmax);
```

```
tol = 1e-10;

pv = [0,0; 5,0; 5,1; 0,1; 0,0];

for nref = 0:(nrefmax - 1)
    [p, t, e] = pmesh(pv, hmax, nref);
    [ein, eout, ewall] = waveguide_edges(p, t);
    [K, M, Bin, Bout, bin] = femhelmholtz(p, t, ein, eout);
    Kk = K - (wave.^2) .* M + imag .* wave .* (Bin + Bout);
    Ff = bin .* 2 .* imag .* wave;
    a = Kk\Ff;

    u_exact = cos(wave .* p(:,1));

    tplot(p, t, real(a) - u_exact)
    k = 1;
    error = [];

    % loop over all the points in the ref soln to find the maximum error
    for i = 1:length(p)
        error(k) = abs(real(a(i)) - u_exact(i));
        k = k + 1;
    end

    % determine the max-norm (maximum element in error())
    errors(nref + 1) = max(error);
end

sizes = 1./hmax .^ (0:(nrefmax - 1));
loglog(sizes, errors, 'o-')
xlabel('1/hmax', 'FontSize', 16)
ylabel('Max-Norm_of_the_Error', 'FontSize', 16)
grid on

for i = 1:(length(errors) - 1)
    rate(i) = log2(errors(i)) - log2(errors(i+1));
end

saveas(gcf, 'error', 'png')
```

5.2 Question 3

5.2.1 Parts (a), (b), and (c) - q3.m

```
% mesh domain
pv = [0,0; 5,0; 5,1; 3.1,1; 3.1,0.2; 2.9,0.2; 2.9,1; 2.1,1; 2.1,0.2; 1.9,0.2;
    ↪ 1.9,1.0; 0,1; 0,0];
[p, t, e] = pmesh(pv, 0.2, 2);

% find the mesh edges
[ein, eout, ewall] = waveguide_edges(p, t);

H = [];
```

```
Wave = 6:0.01:6.5;

% solve femhelmholtz
for wave = Wave

    [K, M, Bin, Bout, bin] = femhelmholtz(p, t, ein, eout, wave);
    Kk = K - (wave.^2) .* M + 1i .* wave .* (Bin + Bout);
    Ff = bin .* 2 .* 1i .* wave;
    a = Kk\Ff;

    % compute complex conjugate transpose
    a_h = transpose(conj(a));

    % compute H(u)
    H = [H, a_h * Bout * a];
end

semilogy(Wave, real(H), 'o-')
xlabel('k')
ylabel('log(H(u))')
grid on

% find maximum and minimum values of H
Hmin = 1;
Hmax = 1;

for i = 1:length(H)
    if H(i) < H(Hmin)
        Hmin = i;
    end
    if H(i) > H(Hmax)
        Hmax = i;
    end
end

% plot solution for minimum H
[K, M, Bin, Bout, bin] = femhelmholtz(p, t, ein, eout, Wave(Hmin));
Kk = K - (Wave(Hmin).^2) .* M + 1i .* Wave(Hmin) .* (Bin + Bout);
Ff = bin .* 2 .* 1i .* Wave(Hmin);
a = Kk\Ff;
tplot(p, t, real(a))

% plot solution for maximum H
[K, M, Bin, Bout, bin] = femhelmholtz(p, t, ein, eout, Wave(Hmax));
Kk = K - (Wave(Hmax).^2) .* M + 1i .* Wave(Hmax) .* (Bin + Bout);
Ff = bin .* 2 .* 1i .* Wave(Hmax);
a = Kk\Ff;
tplot(p, t, real(a))
```

5.3 Question 4

5.3.1 Part (a) - `p2mesh.m`

```
function [p2, t2, e2] = p2mesh(p, t)

e = boundary_nodes(t);

% tplot(p, t)
% hold on

T = zeros(length(t(:,1)), 6);
pnew = []; enew = [];

for i = 1:length(t(:,1))
    A = [p(t(i, 1), 1), p(t(i, 1), 2)];
    B = [p(t(i, 2), 1), p(t(i, 2), 2)];
    C = [p(t(i, 3), 1), p(t(i, 3), 2)];

    pnew = [pnew; (A(1) + B(1))/2, (A(2) + B(2))/2];
    pnew = [pnew; (A(1) + C(1))/2, (A(2) + C(2))/2];
    pnew = [pnew; (C(1) + B(1))/2, (C(2) + B(2))/2];

    % create new row in T for the triangle - save original points
    T(i, 1) = t(i, 1);
    T(i, 3) = t(i, 2);
    T(i, 5) = t(i, 3);
end

pnew = unique(pnew, 'rows');
p2 = [p; pnew];
%scatter(p2(:,1), p2(:,2), 'bo')
%hold on

for i = 1:length(t(:,1))
    A = [p(t(i, 1), 1), p(t(i, 1), 2)];
    B = [p(t(i, 2), 1), p(t(i, 2), 2)];
    C = [p(t(i, 3), 1), p(t(i, 3), 2)];

    pt1 = [(A(1) + B(1))/2, (A(2) + B(2))/2];
    pt2 = [(C(1) + B(1))/2, (C(2) + B(2))/2];
    pt3 = [(A(1) + C(1))/2, (A(2) + C(2))/2];

    T(i, 2) = row_number(pt1, p2);
    T(i, 4) = row_number(pt2, p2);
    T(i, 6) = row_number(pt3, p2);

    for j = 1:length(e)
        if e(j) == T(i, 1) % the 1-coordinate is on the boundary
            for k = 1:length(e)
                if e(k) == T(i, 3) % either 2 or 3-coordinate on boundary
                    enew = [enew; row_number(pt1, p2)];
                end
                if e(k) == T(i, 5)
                    enew = [enew; row_number(pt3, p2)];
                end
            end
        end
    end

    % check if side 2-3 is on the boundary (both nodes in e)
    for j = 1:length(e)
        if e(j) == T(i, 3) % the 2-coordinate is on the boundary
            for k = 1:length(e)
                if e(k) == T(i, 5) % the 3-coordinate is on the boundary
                    enew = [enew; row_number(pt2, p2)];
                end
            end
        end
    end
end

enew = unique(enew);
t2 = T;

% delete points in e2 that appear more than one time in the triangulation
enewer = [];
for k = 1:length(enew)
    flag = 0;

    for i = 1:length(T(:,1))
        for j = 1:6
            if T(i, j) == enew(k)
                flag = flag + 1;
            end
        end
    end

    if flag >= 2
        % point isn't actually on boundary
    elseif flag == 1
        enewer = [enewer; enew(k)];
    end
end

e2 = [e; enewer];

% for i = 1:length(e2)
%     scatter(p2(e2(i), 1), p2(e2(i), 2), 'ro')
%     hold on
% end

disp('Finished generating quadratic mesh...')
end
```

5.3.2 Part (b) - `fempoi2.m`

```
function [a, K, F] = fempoi2(p, t, e)
% p = p in mesh
% t = triangulation
% e = Dirichlet boundary nodes
% --- To use this function, you must have already run p2mesh()

num_elem = length(t(:,1));
num_nodes_per_elem = 6;          % quadratic triangular elements
num_nodes = length(p(:,1));

% form the permutation matrix for assembling the global matrices
[perm] = permutation(num_nodes_per_elem);

% two-point rule from last time (already multiplied by area)
wt = 0.5.*[1/3; 1/3; 1/3];
qp = [1/6,1/6; 2/3,1/6; 1/6,2/3];

% assemble the elemental k and elemental f
K = zeros(num_nodes);
F = zeros(num_nodes, 1);

for elem = 1:num_elem
    k = 0;
    f = 0;

    for ll = 1:length(wt)
        [N, dN_dxe, dN_deta, x_xe_eta, y_xe_eta, dx_dxe, dx_deta, dy_dxe,
            ↪ dy_deta, B] = shapefunctions(qp(ll, 1), qp(ll, 2),
            ↪ num_nodes_per_elem, p, t, elem);
        F_mat = transpose([dx_dxe, dx_deta; dy_dxe, dy_deta]);
        J = det(F_mat);

        % assemble the (elemental) forcing vector
        f = f + wt(ll) * transpose(N) * J;

        % assemble the (elemental) stiffness matrix
        k = k + wt(ll) * transpose(inv(F_mat) * B) * inv(F_mat) * B * J;
    end

    % place the elemental k matrix into the global K matrix
    for m = 1:length(perm(:,1))
       i = perm(m,1);
       j = perm(m,2);
       K(t(elem, i), t(elem, j)) = K(t(elem, i), t(elem, j)) + k(i,j);
    end

    % place the elemental f matrix into the global F matrix
    for i = 1:length(f)
       F(t(elem, i)) = F((t(elem, i))) + f(i);
    end
end

% apply Dirichlet conditions
for i = 1:length(e)
    K(e(i),:) = 0;
    K(e(i),e(i)) = 1;
    F(e(i)) = 0;
end
```

```
a = K\F;
end
```

5.3.3 Part (b) - `convergence_fempoi.m`

```
% script for convergence study of fempoi2
clear all
pv = [0,0; 1,0; 1,1; 0,1; 0,0];
hmax = 0.3;
nrefmax = 4;

errors = zeros(1, nrefmax);

% create initial (very fine) mesh
[p_ref, t, e] = pmesh(pv, hmax, nrefmax);
[p2, t2, e2] = p2mesh(p_ref, t);

% compute the reference solution
[a_ref] = fempoi2(p2, t2, e2);
tplot(p2, t2, a_ref)

for nref = 0:(nrefmax - 1)
    [p, t, e] = pmesh(pv, hmax, nref);
    [p, t, e] = p2mesh(p, t);
    [a] = fempoi2(p, t, e);

    if nref == 0
        P = length(p);
    end

    k = 1;
    error = max(abs(a_ref(1:P) - a(1:P)));

    % determine the max-norm (maximum element in error())
    errors(nref + 1) = error;
end

loglog(1./hmax .^ (0:(nrefmax - 1)), errors, 'o-')
xlabel('log(1/hmax)')
ylabel('Max-Norm of the Error')
grid on

for i = 1:(length(errors) - 1)
    rate(i) = log2(errors(i)) - log2(errors(i+1));
end
```