MATH 228b: HW1

April Novak

February 1, 2017

1

Transfinite Interpolation (TFI) is used to map from a master domain (defined in 2-D over $0 \leq \xi \leq 1$ and $0 \leq \eta \leq 1$) to the physical domain by expressing the boundary of the physical domain in terms of ξ, η coordinates. This then constrains the master grid to lie within the physical domain boundaries. TFI does not guarantee a one-to-one mapping or orthogonality of the mesh, however, but is useful for directly controlling the grid spacing in a computationally efficient and easy manner. The mapping from the master domain to the physical domain is given by $\mathbf{X}(\xi, \eta)$:

$$\mathbf{X}(\xi,\eta) = \left[x(\xi,\eta), y(\xi,\eta)\right]^T \tag{1}$$

where $x(\xi,\eta)$ and $y(\xi,\eta)$ are the mappings from the individual coordinates in the master domain (ξ,η) to those in the physical domain (x,y). For *linear* TFI, we construct 1-D, *linear* interpolants in the x and y directions:

$$\begin{aligned}\mathbf{U}(\xi,\eta) =&(1-\xi_i)\mathbf{X}(0,\eta_j) + \xi_i\mathbf{X}(1,\eta_j)\\ \mathbf{V}(\xi,\eta) =&(1-\eta_j)\mathbf{X}(\xi_i,0) + \eta_j\mathbf{X}(\xi_i,1)\end{aligned} \tag{2}$$

where $\mathbf{U}$ is the 1-D interpolant in the x-direction and $\mathbf{V}$ the 1-D interpolant in the y-direction. For a 2-D domain, x and y are functions of ξ and η. The 2-D interpolant is constructed from the 1-D interpolants above by performing a Boolean sum.

$$\begin{aligned}\mathbf{X}(\xi,\eta) =&(1-\xi_i)\mathbf{X}(0,\eta_j) + \xi_i\mathbf{X}(1,\eta_j) + (1-\eta_j)\mathbf{X}(\xi_i,0) + \eta_j\mathbf{X}(\xi_i,1) -\\ &\left[(1-\xi_i)(1-\eta_j)\mathbf{X}(0,0) + (1-\xi_i)\eta_j\mathbf{X}(0,1) + \xi_i(1-\eta_j)\mathbf{X}(1,0) + \xi_i\eta_j\mathbf{X}(1,1)\right]\end{aligned} \tag{3}$$

The $\mathbf{X}$ that appear on the right-hand side of the equation above are assumed to be known, and is how the physical domain boundary enters the meshing algorithm. For a straight-sided domain, with corner coordinates $(x_1,y_1),(x_2,y_2),(x_3,y_3),(x_4,y_4)$, where the first coordinate pair is in the lower left corner, the interpolation along the sides is given by:

$$\begin{aligned}\mathbf{X}(0,\eta_j) =& \begin{bmatrix}(1-\eta_j)x_1 + \eta x_4\\(1-\eta_j)y_0 + \eta y_4\end{bmatrix}\\ \mathbf{X}(1,\eta_j) =& \begin{bmatrix}(1-\eta_j)x_2 + \eta x_3\\(1-\eta_j)y_2 + \eta y_3\end{bmatrix}\\ \mathbf{X}(\xi_i,0) =& \begin{bmatrix}(1-\xi_i)x_1 + \xi_i x_2\\(1-\xi_i)y_1 + \xi_i y_2\end{bmatrix}\\ \mathbf{X}(\xi_i,1) =& \begin{bmatrix}(1-\xi_i)x_4 + \xi_i x_3\\(1-\xi_i)y_4 + \xi_i y_3\end{bmatrix}\end{aligned} \tag{4}$$

where a 2-D domain is used for simplicity - the results would equally extend to 3-D. Then, inserting these into the Boolean sum above gives:

$$\begin{aligned}
x(\xi,\eta) =&(1-\xi_i)\big((1-\eta_j)x_1+\eta_j x_4\big)+\xi_i\big((1-\eta_j)x_2+\eta x_3\big)+\\
&(1-\eta_j)\big((1-\xi_i)x_1+\xi_i x_2\big)+\eta_j\big((1-\xi_i)x_4+\xi_i x_3\big)-\\
&\big[(1-\xi_i)(1-\eta_j)x_1+(1-\xi_i)\eta_j x_4+\xi_i(1-\eta_j)x_2+\xi_i\eta_j x_3\big]\\
=&\cancel{(1-\xi_i)(1-\eta_j)x_1}+\cancel{(1-\xi_i)\eta_j x_4}+\cancel{\xi_i(1-\eta_j)x_2}+\cancel{\eta_j x_3\xi_i}+\\
&(1-\eta_j)(1-\xi_i)x_1+\xi_i x_2(1-\eta_j)+\eta_j(1-\xi_i)x_4+\xi_i x_3\eta_j-\\
&\big[\cancel{(1-\xi_i)(1-\eta_j)x_1}+\cancel{(1-\xi_i)\eta_j x_4}+\cancel{\xi_i(1-\eta_j)x_2}+\cancel{\xi_i\eta_j x_3}\big]\\
=&(1-\eta_j)(1-\xi_i)x_1+\xi_i x_2(1-\eta_j)+\eta_j(1-\xi_i)x_4+x_3\eta_j\xi_i
\end{aligned} \tag{5}$$

From the last line above, it is seen that, for the transformation from ξ to x, the linear TFI is equivalent to bilinear interpolation between the four corner points. Similarly, bilinear interpolation between the y-coordinates of the four corner points is also equivalent to a bilinear interpolation (work follows the same steps shown above).

$$y(\xi,\eta)=(1-\eta_j)(1-\xi_i)y_1+\xi_i y_2(1-\eta_j)+\eta_j(1-\xi_i)y_4+y_3\eta_j\xi_i \tag{6}$$

2

TFI is used to generate a structured mesh of the given domain. The 1-D interpolations in the x and y directions now include four terms, two to account for control of the 1-D boundary values, and two to control for the 1-D boundary derivatives.

$$\begin{aligned}
\mathbf{U}(\xi,\eta)&=(2\xi_i^3-3\xi_i^2+1)\mathbf{X}(0,\eta_j)+(3\xi_i^2-2\xi_i^3)\mathbf{X}(1,\eta_j)+(\xi_i^3-2\xi_i^2+\xi_i)\frac{\partial\mathbf{X}(0,\eta_j)}{\partial\xi}+(\xi_i^3-\xi_i^2)\frac{\partial\mathbf{X}(1,\eta_j)}{\partial\xi}\\
\mathbf{V}(\xi,\eta)&=(2\eta_j^3-3\eta_j^2+1)\mathbf{X}(\xi_i,0)+(3\eta_j^2-2\eta_j^3)\mathbf{X}(\xi_i,1)+(\eta_j^3-2\eta_j^2+\eta_j)\frac{\partial\mathbf{X}(\xi_i,0)}{\partial\eta}+(\eta_j^3-\eta_j^2)\frac{\partial\mathbf{X}(\xi_i,1)}{\partial\eta}
\end{aligned} \tag{7}$$

The 2-D interpolant is constructed from the 1-D interpolants by performing a Boolean sum:

$$\begin{aligned}\mathbf{X}(\xi,\eta) =&(2\xi_i^3-3\xi_i^2+1)\mathbf{X}(0,\eta_j)+(3\xi_i^2-2\xi_i^3)\mathbf{X}(1,\eta_j)+(\xi_i^3-2\xi_i^2+\xi_i)\frac{\partial\mathbf{X}(0,\eta_j)}{\partial\xi}+(\xi_i^3-\xi_i^2)\frac{\partial\mathbf{X}(1,\eta_j)}{\partial\xi}+\\
&(2\eta_j^3-3\eta_j^2+1)\mathbf{X}(\xi_i,0)+(3\eta_j^2-2\eta_j^3)\mathbf{X}(\xi_i,1)+(\eta_j^3-2\eta_j^2+\eta_j)\frac{\partial\mathbf{X}(\xi_i,0)}{\partial\eta}+(\eta_j^3-\eta_j^2)\frac{\partial\mathbf{X}(\xi_i,1)}{\partial\eta}-\\
&\Big[(2\xi_i^3-3\xi_i^2+1)(2\eta_j^3-3\eta_j^2+1)\mathbf{X}(0,0)+(2\xi_i^3-3\xi_i^2+1)(3\eta_j^2-2\eta_j^3)\mathbf{X}(0,1)\Big]-\\
&\left[(2\xi_i^3-3\xi_i^2+1)(\eta_j^3-2\eta_j^2+\eta_j)\frac{\partial\mathbf{X}(0,0)}{\partial\eta}+(2\xi_i^3-3\xi_i^2+1)(\eta_j^3-\eta_j^2)\frac{\partial\mathbf{X}(0,1)}{\partial\eta}\right]-\\
&\Big[(3\xi_i^2-2\xi_i^3)(2\eta_j^3-3\eta_j^2+1)\mathbf{X}(1,0)+(3\xi_i^2-2\xi_i^3)(3\eta_j^2-2\eta_j^3)\mathbf{X}(1,1)\Big]-\\
&\left[(3\xi_i^2-2\xi_i^3)(\eta_j^3-2\eta_j^2+\eta_j)\frac{\partial\mathbf{X}(1,0)}{\partial\eta}+(3\xi_i^2-2\xi_i^3)(\eta_j^3-\eta_j^2)\frac{\partial\mathbf{X}(1,1)}{\partial\eta}\right]-\\
&\left[(\xi_i^3-2\xi_i^2+1)(2\eta_j^3-3\eta_j^2+1)\frac{\partial\mathbf{X}(0,0)}{\partial\xi}+(\xi_i^3-2\xi_i^2+\xi_i)(3\eta_j^2-2\eta_j^3)\frac{\partial\mathbf{X}(0,1)}{\partial\xi}\right]-\\
&\left[(\xi_i^3-2\xi_i^2+1)(\eta_j^3-2\eta_j^2+\eta_j)\frac{\partial\mathbf{X}(0,0)}{\partial\xi\partial\eta}+(\xi_i^3-2\xi_i^2+\xi_i)(\eta_j^3-\eta_j^2)\frac{\partial\mathbf{X}(0,1)}{\partial\xi\partial\eta}\right]-\\
&\left[(\xi_i^3-\xi_i^2)(2\eta_j^3-3\eta_j^2+1)\frac{\partial\mathbf{X}(1,0)}{\partial\xi}+(\xi_i^3-\xi_i^2)(3\eta_j^2-2\eta_j^3)\frac{\partial\mathbf{X}(1,1)}{\partial\xi}\right]-\\
&\left[(\xi_i^3-\xi_i^2)(\eta_j^3-2\eta_j^2+\eta_j)\frac{\partial\mathbf{X}(1,0)}{\partial\xi\partial\eta}+(\xi_i^3-\xi_i^2)(\eta_j^3-\eta_j^2)\frac{\partial\mathbf{X}(1,1)}{\partial\xi\partial\eta}\right]
\end{aligned} \tag{8}$$

For the given domain, the $\mathbf{X}$ functions along the sides are given by:

$$\begin{aligned}\mathbf{X}(\xi,0)&=\big[\xi,(1-cos(2\pi\xi))/5\big]^T\\
\mathbf{X}(\xi,1)&=\big[\xi,1+(1-cos(\pi\xi))/5\big]^T\\
\mathbf{X}(0,\eta)&=[0,\eta]^T\\
\mathbf{X}(1,\eta)&=[1,1.4\eta]^T\end{aligned} \tag{9}$$

where the values at the corners are:

$$\begin{aligned}\mathbf{X}(0,0)&=[0,0]^T\\
\mathbf{X}(0,1)&=[0,1]^T\\
\mathbf{X}(1,0)&=[1,0]^T\\
\mathbf{X}(1,1)&=[1,1.4]^T\end{aligned} \tag{10}$$

And the normals along the boundaries, to be used in the specification of the derivatives according to the problem statement, are:

$$\begin{aligned}\hat{n}(\text{bottom side})=&\frac{\big[-2\pi sin(2\pi\xi)/5,1\big]^T}{\sqrt{(-2\pi sin(2\pi\xi)/5)^2+1}}\\
\hat{n}(\text{top side})=&\frac{\big[-\pi sin(\pi\xi)/5,1\big]^T}{\sqrt{(-\pi sin(\pi\xi)/5)^2+1}}\\
\hat{n}(\text{left side})=&[1,0]^T\\
\hat{n}(\text{right side})=&[1,0]^T\end{aligned} \tag{11}$$

Fig. 1 shows the mesh generated using $T=0.5$. As can be seen, boundary orthogonality is achieved on all sides. For simplicity, the second derivative terms were all assumed to be zero, since they would be zero on the left and right edges, and all corners points can be expressed as either belonging to the right and left edges, or either to the top and bottom edges. The code used for this problem is given in the Appendix.

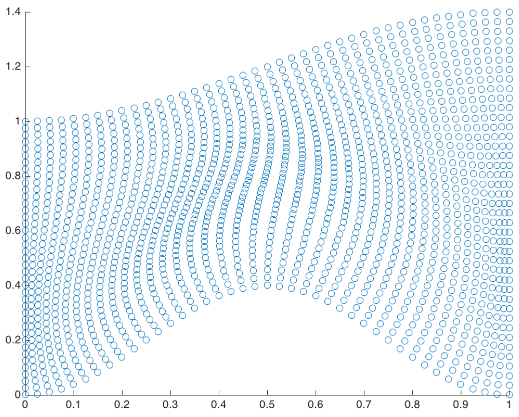


Figure 1. Hermite mapping.

3

For a conformal map, an analytic function can be mapped to the domain $w = u + iv$ for a variable $z = x + iy$.

$$\begin{aligned} w =& \frac{2e^z - 3}{3e^z - 2} \\ =& \frac{2e^{x+iy} - 3}{3e^{x+iy} - 2} \end{aligned} \tag{12}$$

Then, multiply this expression by the complex conjugate of the denominator to find how x, y maps to u, v.

$$\begin{aligned} w =& \left(\frac{2e^{x+iy} - 3}{3e^{x+iy} - 2}\right)\left(\frac{3e^{x-iy} - 2}{3e^{x-iy} - 2}\right) \\ =& \frac{6e^{x+iy}e^{x-iy} - 4e^{x+iy} - 9e^{x-iy} + 6}{9e^{x+iy}e^{x-iy} - 6e^{x+iy} + -6e^{x-iy} + 4} \\ =& \frac{6e^{2x} - 13e^x cos(y) + 5e^x isin(y) + 6}{9e^{2x} - 12e^x cos(y) + 4} \\ =& \frac{6e^{2x} - 13e^x cos(y) + 6}{9e^{2x} - 12e^x cos(y) + 4} + i\frac{5e^x sin(y)}{9e^{2x} - 12e^x cos(y) + 4} \end{aligned} \tag{13}$$

x, y maps to u, v based on the following expressions.

$$\begin{aligned} u =& \frac{6e^{2x} - 13e^x cos(y) + 6}{9e^{2x} - 12e^x cos(y) + 4} \\ v =& \frac{5e^x sin(y)}{9e^{2x} - 12e^x cos(y) + 4} \end{aligned} \tag{14}$$

Fig. 2 shows the mesh generated by uniformly discretizing x into 21 values and y into 81 values, and then using the above conformal mapping to determine the coordinates in the u, v coordinate system. The code written for this section is given in the Appendix.

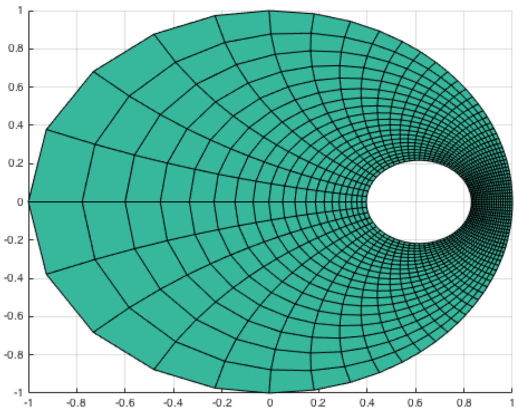


Figure 2. Conformal mapping structured mesh in the u, v coordinate system.

This transformation can also be shown by looking at the transformation values along the edges of the x, y domain by substituting in the appropriate values of x and y into Eq. (14). u transforms according to:

$$\begin{aligned}
\text{Along left edge } (x = 0, y = y) : u =& \frac{12 - 13cos(y)}{13 - 12cos(y)} \\
\text{Along right edge } (x = 1, y = y) : u =& \frac{(6e^2 + 6) - 13ecos(y)}{(9e^2 + 4) - 12ecos(y)} \\
\text{Along bottom edge } (x = x, y = 0) : u =& \frac{6e^{2x} - 13e^x + 6}{9e^{2x} - 12e^x + 4} \\
\text{Along top edge } (x = x, y = 2\pi) : u =& \frac{6e^{2x} - 13e^x + 6}{9e^{2x} - 12e^x + 4}
\end{aligned} \tag{15}$$

v transforms according to:

$$\begin{aligned}
\text{Along left edge } (x = 0, y = y) : v =& \frac{5sin(y)}{13 - 12cos(y)} \\
\text{Along right edge } (x = 1, y = y) : v =& \frac{5esin(y)}{(9e^2 + 4) - 12ecos(y)} \\
\text{Along bottom edge } (x = x, y = 0) : v =& 0 \\
\text{Along top edge } (x = x, y = 2\pi) : v =& 0
\end{aligned} \tag{16}$$

Because u transforms the same way for the bottom and top edges, the bottom and top edges of the x, y domain "touch" in the u, v domain. Because $0 \leq x \leq 1$, and $v = 0$ along the bottom and top edges, the bottom and top edges in the master domain map to the line segment between $-1 \leq u \leq (6e^2 - 13e + 6)/(9e^2 - 12e + 4) \approx 0.395$.

The left side of the master domain gets mapped to the "outside" of the conformal map, while the right side of the physical domain is mapped to the perimeter of the inner circle. Along the left side of the master domain, $x = 0$ and $y = y$, giving the following transformations:

$$u = \frac{12 - 13cos(y)}{13 - 12cos(y)} v = \quad \frac{5sin(y)}{13 - 12cos(y)} \tag{17}$$

For a circle, the general equation is $u^2 + v^2 = r^2$, where r is the radius of the circle. By computing the left-hand side of this expression, it is shown that the outer boundary of the conformal map is a circle with radius 1, which is observed in the mesh shown above. With the $0, 0$ coordinate mapping to $-1, 0$ as given by the above expressions, this shows how the outer boundary of the conformal map is determined.

$$
\begin{aligned}
u^2 + v^2 &= \frac{(12 - 13cos(y))^2}{(13 - 12cos(y))^2} + \frac{(5sin(y))^2}{(13 - 12cos(y))^2} \\
&= \frac{144 - 312cos(y) + 169cos^2(y) + 25sin^2(y)}{169 - 312cos(y) + 144cos^2(y)} \\
&= \frac{144 - 312cos(y) + 25(cos^2(y) + sin^2(y)) + 144cos^2(y)}{169 - 312cos(y) + 144cos^2(y)} \\
&= \frac{169 - 312cos(y) + 144cos^2(y)}{169 - 312cos(y) + 144cos^2(y)} \\
&= 1
\end{aligned}
\tag{18}
$$

For the inner circle, because $v = 0$ on the edge of the circle, the v coordinate is not offset from the origin, so the equation for this circle is given by:

$$
(u - u_0)^2 + v^2 = r^2 \tag{19}
$$

where $(u_0, 0)$ is the center of the small circle. The diameter of the circle is given by the difference in u between the left edge of the circle (corresponding to the $(x = 1, y = 0)$ point in the master domain) and the right edge of the circle (corresponding to the $(x = 1, y = \pi)$ point in the master domain).

$$
r = \frac{1}{2}\left(\frac{(6e^2 + 6) + 13e}{(9e^2 + 4) + 12e} - \frac{(6e^2 + 6) - 13e}{(9e^2 + 4) - 12e}\right) = \frac{5e}{9e^2 - 4} \tag{20}
$$

Hence, to show that the right edge maps to the inner circle, the equation for the circle is computed in order to find the radius above:

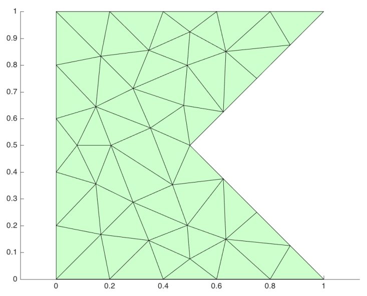
$$
\left(\frac{6e^2 + 6 - 13ecos(y)}{9e^2 + 4 - 12ecos(y)} - \left(\frac{(6e^2 + 6) - 13e}{(9e^2 + 4) - 12e} + \frac{5e}{9e^2 - 4}\right)\right)^2 + \left(\frac{5esin(y)}{b - 12ecos(y)}\right)^2 = r^2 \tag{21}
$$

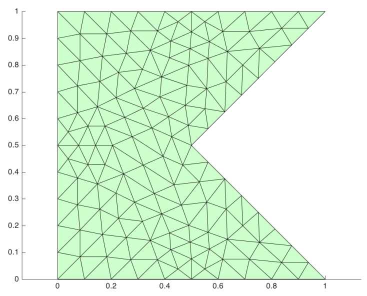
Because this algebra would be very messy, it is not carried out here, but performing the algebra above would show that r is indeed given by Eq. (20), showing that the mapping does map the right edge to the inner circle.

4

Delaunay triangulation is used to generate an unstructured, triangular mesh of the given polygonal domain. For a given maximum element size, the boundary is discretized such that each element has a length as close as possible to h_{max}, but not exceeding this value. Then, the domain is triangulated using Delaunay triangulation. Triangles outside the domain are deleted by placing a point at the midpoint of each side of every triangle, and then testing to see if that point is outside the domain using the `inpolygon` command. This is a relatively simple method of determining whether a line is outside the polygon, and will only work for relatively simple geometries. For instance, if only a portion of a line extends outside the domain, then this procedure might not work, since the midpoint of the line might be inside the domain, but the entire line may not be inside the domain. This would require a constrained Delaunay formulation, which is beyond the scope of this assignment.

Triangles with a side outside the domain are deleted, and the triangulation process repeated until all triangles have a size no larger than $h_{max}^2/2$. Triangles with areas larger than this maximum value are considered to be poor triangles from a mesh quality standpoint, and their circumcenter is added. This algorithm is not implemented in a very robust manner, however, since that circumcenter must be in the domain for the refinement process to continue to completion, since if the circumcenter is outside the domain, then that point is deleted, and an infinite loop is entered. So, this is a second reason why my implementation only works for relatively simple geometries. Once all triangles have an area smaller than the maximum allowable area, the mesh if refined uniformly n_{ref} times. For each refinement, the center of each mesh edge is added, and the triangulation performed to get a roughly twice-as-fine mesh as the previous uniform refinement level. For the polygon given, Fig. 3 shows the Delaunay triangulation obtained for (a) zero, (b) one, and (c) two uniform refinements for $h_{max} = 0.2$.


(a) $n_{ref} = 0$


(b) $n_{ref} = 1$

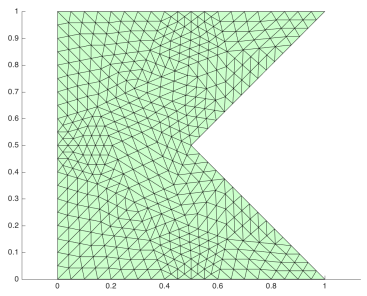
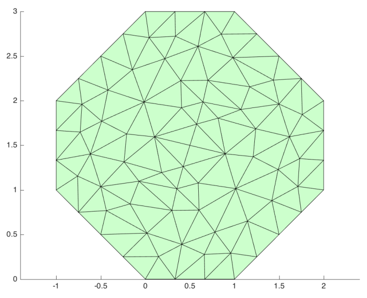
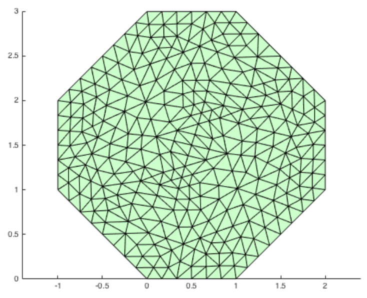

(c) $n_{ref} = 2$

Figure 3. Uniform refinement for the starting polygon.

Two more examples of different polygons are shown in Figs. 4 and 5.


(a) $n_{ref} = 0$


(b) $n_{ref} = 1$

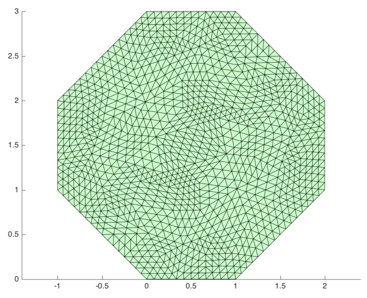
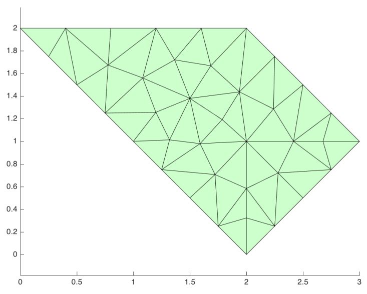
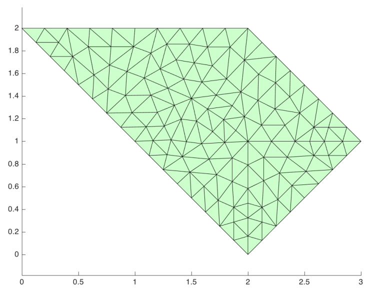

(c) $n_{ref} = 2$

Figure 4. Uniform refinement for different polygon, for $h_{max} = 4$.


(a) $n_{ref} = 0$


(b) $n_{ref} = 1$

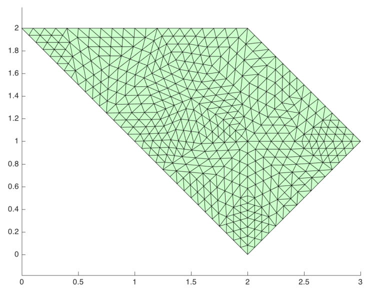

(c) $n_{ref} = 2$

Figure 5. Uniform refinement for different polygon, for $h_{max} = 4$.

Finally, the `boundary_nodes` function is used to find the indices of all the nodes on the boundary, which is required for later numerical methods such as the finite element method. All of the code used for this problem is included in the Appendix and submitted online.

5 Appendix

5.1 Question 2 Code

5.1.1 `hermite_map.m`

This function plots the hermite map in Question 2.

```
% Question 2, MATH-228b HW 1
clear all

T = 0.5;

a1 = @(xi) 2*xi^3 - 3*xi^2 + 1;
a2 = @(xi) 3*xi^2 -2*xi^3;
a3 = @(xi) xi^3 - 2*xi^2 + xi;
a4 = @(xi) xi^3 - xi^2;

b1 = @(eta) 2*eta^3 - 3*eta^2 + 1;
b2 = @(eta) 3*eta^2 -2*eta^3;
b3 = @(eta) eta^3 - 2*eta^2 + eta;
b4 = @(eta) eta^3 - eta^2;

leftx =      @(xi,eta) 0;
lefty =      @(xi,eta) eta;
rightx =     @(xi,eta) 1;
righty =     @(xi,eta) 1.4*eta;
bottomx =    @(xi,eta) xi;
bottomy =    @(xi,eta) (1-cos(2*pi*xi))/5;
topx =       @(xi,eta) xi;
topy =       @(xi,eta) 1 + (1-cos(pi*xi))/5;

leftnx =     @(xi,eta) 1;
leftny =     @(xi,eta) 0;
rightnx =    @(xi,eta) 1;
rightny =    @(xi,eta) 0;

bottomnx = @(xi,eta) (-2*pi*sin(2*pi*xi)/5)/sqrt((-2*pi*sin(2*pi*xi)/5)^2+1);
bottomny = @(xi,eta) (1)/sqrt((-2*pi*sin(2*pi*xi)/5)^2+1);
topnx = @(xi,eta) (-pi*sin(pi*xi)/5)/sqrt((-pi*sin(pi*xi)/5)^2+1);
topny = @(xi,eta) (1)/sqrt((-pi*sin(pi*xi)/5)^2+1);

k = 1;
for dxi = linspace(0, 1, 41)
    for deta = linspace(0, 1, 41)
        x(k) = a1(dxi)*leftx(dxi, deta) + a2(dxi)*rightx(dxi, deta) + b1(deta)
            ↪ *bottomx(dxi, deta) + b2(deta)*topx(dxi, deta) ... % original
            ↪ four terms
          + a3(dxi)*T*leftnx(dxi,deta) + a4(dxi)*T*rightnx(dxi,deta) + b3(
              ↪ deta)*T*bottomnx(dxi,deta) + b4(deta)*T*topnx(dxi,deta) ...
              ↪ % four derivative terms
          - (a1(dxi)*b1(deta)*leftx(0, 0) + a1(dxi)*b2(deta)*leftx(0, 1) +
              ↪ a2(dxi)*b1(deta)*rightx(1,0) + a2(dxi)*b2(deta)*rightx
              ↪ (1,1)) ... % subtract out corners
          - (a1(dxi)*b3(deta)*T*bottomnx(0,0)   + a1(dxi)*b4(deta)*T*topnx
                    ↪ (0,1))   ...
                  - (a2(dxi)*b3(deta)*T*bottomnx(1,0)    + a2(dxi)*b4(deta)*T*topnx
                    ↪ (1,1))   ...
                  - (b1(deta)*a3(dxi)*T*leftnx(0,0)      + b1(deta)*a4(dxi)*T*
                    ↪ rightnx(1,0))   ...
                  - (b2(deta)*a3(dxi)*T*leftnx(0,1)      + b2(deta)*a4(dxi)*T*
                    ↪ rightnx(1,1));

        y(k) = a1(dxi)*lefty(dxi, deta) + a2(dxi)*righty(dxi, deta) + b1(deta)
            ↪ *bottomy(dxi, deta) + b2(deta)*topy(dxi, deta) ... % original
            ↪ four terms
              + a3(dxi)*T*leftny(dxi,deta) + a4(dxi)*T*rightny(dxi,deta) + b3(
                  ↪ deta)*T*bottomny(dxi,deta) + b4(deta)*T*topny(dxi,deta) ...
                  ↪  % four derivative terms
             - (a1(dxi)*b1(deta)*lefty(0, 0) + a1(dxi)*b2(deta)*lefty(0, 1) +
                 ↪ a2(dxi)*b1(deta)*righty(1,0) + a2(dxi)*b2(deta)*righty(1,1))
                 ↪  ... % subtract out corners
             - (a1(dxi)*b3(deta)*T*bottomny(0,0)     + a1(dxi)*b4(deta)*T*topny
                 ↪ (0,1)) ...
             - (a2(dxi)*b3(deta)*T*bottomny(1,0)     + a2(dxi)*b4(deta)*T*topny
                 ↪ (1,1)) ...
             - (b1(deta)*a3(dxi)*T*leftny(0,0)       + b1(deta)*a4(dxi)*T*
                 ↪ rightny(1,0)) ...
             - (b2(deta)*a3(dxi)*T*leftny(0,1)       + b2(deta)*a4(dxi)*T*
                 ↪ rightny(1,1));
        k = k + 1;
    end
end

scatter(x, y, 'o')
saveas(gcf, 'hermite_plot', 'png')
```

5.2 Question 3 Code

5.2.1 conformal_map.m

This function plots the conformal map in Question 3.

```
% Question 3, MATH 228b HW 1
x = linspace(0, 1, 21);
y = linspace(0, 2*pi, 81);

[X, Y] = meshgrid(x, y);

u = (6.*exp(2.*X)-13.*exp(X).*cos(Y)+6)./(9.*exp(2.*X)-12.*exp(X).*cos(Y)+4);
v = 5.*exp(X).*sin(Y)./(9.*exp(2.*X)-12.*exp(X).*cos(Y)+4);

surf(u, v, zeros.*u)
```

5.3 Question 4 Code

5.3.1 `pmesh.m`

This function is the main function asked for in this problem, and returns the node coordinates, triangulation, and boundary nodes. This function calls several sub-functions that were developed, which are described in the next sections.

```
function [p, t, e] = pmesh(pv, hmax, nref)
% Uses Delaunay triangulation and refinement to mesh a polygon with
% vertices pv, maximum element side length hmax, and number of uniform
% refinements nref. The return values of this function are the node points
% p, the triangle indices t, and the indices of the boundary points e.

% place points on the domain boundaries according to hmax
[new_pts] = initial_mesh(pv, hmax);

% assemble all of the starting node points
pv_orig = pv(1:end-1, :);
pv = [pv_orig; new_pts; 2,2];

max = hmax^2 / 2;
max_area = max + 1; % dummy initial value
while max_area > max
    % find which points are outside the domain, then delete them
    pv = unique(pv, 'rows');
    [pv] = delete_outside(pv, pv_orig);

    % triangulate the domain
    T = delaunayn(pv);

    % find which triangles are outside the domain, then delete them from T
    [T] = delete_outside_triangles(T, pv, pv_orig);
    tplot(pv, T)

    % compute triangle areas using Heron's formula
    for i = 1:length(T(:,1))
        A = [pv(T(i, 1), 1), pv(T(i, 1), 2)];
        B = [pv(T(i, 2), 1), pv(T(i, 2), 2)];
        C = [pv(T(i, 3), 1), pv(T(i, 3), 2)];
        area(i) = (A(1) * (B(2) - C(2)) + B(1) * (C(2) - A(2)) + C(1) * (A(2)
            ↪ - B(2))) / 2;
        if i == 1
            max_area = area(1);
        end

        if area(i) > max_area
            max_area = area(i);
            refine = i;
        end
    end

    A = [pv(T(refine, 1), 1), pv(T(refine, 1), 2)];
    B = [pv(T(refine, 2), 1), pv(T(refine, 2), 2)];
    C = [pv(T(refine, 3), 1), pv(T(refine, 3), 2)];
    [pt] = circumcenter(A, B, C, pv);
    pv = [pv; pt];
end

% remove last triangulation (not needed since we just broke from loop)
pv = pv(1:(end-1), :);
tplot(pv, T)

% perform uniform refinements
for uf = 1:nref
    sprintf('Performing uniform refinement %i', uf)

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

        % add the three midpoints
        pv = [pv; (A(1) + B(1))/2, (A(2) + B(2))/2];
        pv = [pv; (A(1) + C(1))/2, (A(2) + C(2))/2];
        pv = [pv; (C(1) + B(1))/2, (C(2) + B(2))/2];
    end

    pv = unique(pv, 'rows');
    T = delaunayn(pv);

    % find which triangles are outside the domain, then delete them from T
    [T] = delete_outside_triangles(T, pv, pv_orig);

    tplot(pv, T)
    hold on
end

p = pv;
t = T;
e = boundary_nodes(T);

end
```

5.3.2 `initial_mesh.m`

This function places the initial boundary nodes on the boundary according to the requirement that the element side lengths along the boundary be as close as possible to h_{max}, but not exceed this maximum value.

```
function [new_pts] = initial_mesh(pv, hmax)
% place points on the domain boundaries according to hmax

k = 1;
l = 1;

for i = 1:(size(pv(:,1)) - 1)
    p = 1;
    x0 = pv(i, 1);
    x_next = pv(i + 1, 1);
    y0 = pv(i, 2);
    y_next = pv(i + 1, 2);

    if x_next == x0          % vertical line
        dist = abs(y_next - y0);
        num_divs = ceil(dist/hmax);
        spacing = dist/num_divs;

        for j = 1:(num_divs - 1)
            if y_next > y0
                y = y0 + spacing;
                y0 = y;
            else
                y = y0 - spacing;
                y0 = y;
            end
            new_pts(k, :) = [x0, y];
            k = k + 1;
        end
    else         % not a vertical line
        m(l) = (y_next - y0) / (x_next - x0);

        % determine number of points to place to get close to hmax
        dist = sqrt((y_next - y0)^2 + (x_next - x0)^2);
        num_divs = ceil(dist/hmax);
        spacing = dist/num_divs;

        % compute the new points
        for j = 1:(num_divs - 1)
            if x_next > x0
                x = x0 + sqrt(spacing^2 / (m(l)^2 + 1));
            else
                x = x0 - sqrt(spacing^2 / (m(l)^2 + 1));
            end
            y = m(l) * x - m(l) * x0 + y0;
            x0 = x;
            y0 = y;
            new_pts(k, :) = [x, y];
            k = k + 1;
        end
    end
    l = l + 1;
end
end
```

5.3.3 `delete_outside.m`

This function deletes any points that are outside the domain using the `inpolygon` command.

```
function [pv] = delete_outside(pv, pv_orig)
% delete points outside the domain

j = 1;
```

```
for i = 1:size(pv(:,1))
    in = inpolygon(pv(i,1), pv(i,2), pv_orig(:,1), pv_orig(:,2));

    if in == 0         % generic point outside the domain
    else               % inside the domain
        pv_inside(j,:) = pv(i, :);
        j = j + 1;
    end
end

pv = pv_inside;

end
```

5.3.4 `delete_outside_triangles.m`

This function deletes any triangles that are outside the domain using the `inpolygon` command. A point is placed on each triangle edge at the midpoint, and the `inpolygon` command is used to test whether or not that point is outside the domain. If that point is outside the domain, then the triangle is assumed to be entirely outside the domain, and it is removed from the triangulation.

```
function [T_outside_removed] = delete_outside_triangles(T, pv, pv_orig)
% delete triangles outside the domain

s = 1;
for i = 1:length(T(:,1))
    A = [pv(T(i, 1), 1), pv(T(i, 1), 2)];
    B = [pv(T(i, 2), 1), pv(T(i, 2), 2)];
    C = [pv(T(i, 3), 1), pv(T(i, 3), 2)];
    ptA = [(A(1) + B(1)) / 2, (A(2) + B(2)) / 2];
    ptB = [(A(1) + C(1)) / 2, (A(2) + C(2)) / 2];
    ptC = [(C(1) + B(1)) / 2, (C(2) + B(2)) / 2];
    A_in = inpolygon(ptA(1), ptA(2), pv_orig(:,1), pv_orig(:,2));
    B_in = inpolygon(ptB(1), ptB(2), pv_orig(:,1), pv_orig(:,2));
    C_in = inpolygon(ptC(1), ptC(2), pv_orig(:,1), pv_orig(:,2));

    if A_in && B_in && C_in
        T_outside_removed(s, :) = T(i, :);
        s = s + 1;
    end
end


end
```

5.3.5 `circumcenter.m`

This function computes the circumcenter of three points.

```
function [pt] = circumcenter(A, B, C, pv)

AB = [(A(1) + B(1)) / 2, (A(2) + B(2)) / 2];
m_AB = (B(2) - A(2)) / (B(1) - A(1));
```

```
AC = [(A(1) + C(1)) / 2, (A(2) + C(2)) / 2];
m_AC = (C(2) - A(2)) / (C(1) - A(1));

BC = [(B(1) + C(1)) / 2, (B(2) + C(2)) / 2];
m_BC = (C(2) - B(2)) / (C(1) - B(1));

if m_BC == 0
    m_AB = -1/m_AB;
    m_AC = -1/m_AC;
    mat = [1, -m_AC; 1, -m_AB];
    b = [-m_AC * AC(1) + AC(2); -m_AB * AB(1) + AB(2)];
elseif m_AC == 0
    m_AB = -1/m_AB;
    m_BC = -1/m_BC;
    mat = [1, -m_BC; 1, -m_AB];
    b = [-m_BC * BC(1) + BC(2); -m_AB * AB(1) + AB(2)];
else
    m_AC = -1/m_AC;
    m_BC = -1/m_BC;
    mat = [1, -m_BC; 1, -m_AC];
    b = [-m_BC * BC(1) + BC(2); -m_AC * AC(1) + AC(2)];
end

circumcenter = transpose(mat\b);
pt(1) = circumcenter(2);
pt(2) = circumcenter(1);

end
```

5.3.6 `tplot.m`

This function plots the Delaunay triangulation.

```
function tplot(p, t, u)
%TPLOT Plot triangular mesh P,T and, optionally, solution U

% UC Berkeley Math 228B, Per-Olof Persson <persson@berkeley.edu>

    clf
    if nargin < 3
        patch('vertices',p, 'faces',t, 'facecol',[.8, 1, .8], 'edgecol','k');
    else
        patch('vertices',p, 'faces',t, 'facevertexcdata',u, ...
              'facecol','interp', 'edgecol','none');
        colorbar
    end
    set(gcf, 'renderer','zbuffer');
    axis equal
    drawnow
end
```

5.3.7 `boundary_nodes.m`

This function returns the indices of nodes on the boundary.

```
function e = boundary_nodes(t)
%BOUNDARY_NODES Find boundary nodes E of triangulation T

% UC Berkeley Math 228B, Per-Olof Persson <persson@berkeley.edu>

    edges = [t(:,[1,2]);
             t(:,[2,3]);
             t(:,[3,1])];
    edges = sort(edges, 2);
    [~, ix, jx] = unique(edges, 'rows');
    vec = histc(jx, 1:max(jx));
    qx = find(vec == 1);
    e = edges(ix(qx), :);
    e = unique(e(:));
end
```