MATH 228b: HW2

April Novak

February 16, 2017

Note: The Professor's version of `pmesh.m` is used for this assignment instead of my own from HW 1 because I got points off (I presume) due to my meshing.

1

The solution of the following problem with linear Lagrange elements for $u(0) = 0$ and $u'(1) = g$ is equivalent to a second order central difference approximation of the problem.

$$-u''(x) = f(x) \tag{1}$$

The function space for the weight function v is in the space V_h such that v is C^0 continuous over the entire domain (the zeroth derivative is the highest continuous derivative), where over each element K, v is a linear function. v satisfies the homogeneous form of the Dirichlet boundary condition.

$$V_h = \{v \in C^0([0,1]) : v|_K \in P_1(K) \forall K \in T_h, v(0) = 0\} \tag{2}$$

The weighted residual form is obtained by multiplying Eq. (1) by v and integrating over the domain. The first term on the left-hand side can be rewritten using the divergence rule:

$$-\int_0^1 u''(x)v(x)dx = \int_0^1 f(x)v(x)dx \int_0^1 u'(x)v'(x)dx - \left[u'(x)v(x)\right]_0^1 = \int_0^1 f(x)v(x)dx \tag{3}$$

At this point, the weighted residual form has been reduced to the weak form. The weak form above contains no information about the boundary conditions. By applying the boundary conditions, the above reduces to:

$$\int_0^1 u'(x)v'(x)dx = \int_0^1 f(x)v(x)dx + gv(1) \tag{4}$$

Using a linear Lagrange basis, defined on a mesh with element sizes h, the elements are defined on $e^1 : [0, h]; e^2 : [h, 2h]; e^3 : [3h, 4h]; \cdots ; e^n : [1 - h, h]$, where n is the number of elements. The left and right coordinates of each element will be denoted as x_i and x_{i+1} for simplicity. Then, over an arbitrary element, the linear Lagrange shape functions ψ are:

$$\begin{aligned} \psi_i(x) &= \frac{x_{i+1} - x}{x_{i+1} - x_i} = \frac{x_{i+1} - x}{h} \\ \psi_{i+1}(x) &= \frac{x - x_i}{x_{i+1} - x_i} = \frac{x - x_i}{h} \end{aligned} \tag{5}$$

And their derivatives with respect to x are:

$$\begin{aligned} \psi_i'(x) &= -\frac{1}{h} \\ \psi_{i+1}'(x) &= \frac{1}{h} \end{aligned} \tag{6}$$

For equal-sized elements without edges on the Neumann boundary, the element stiffness matrix and element load vector are given as:

$$\begin{aligned} A_{ij}^k &= \int_{x_i}^{x_{i+1}} \psi_i'(x)v_j'(x)dx \\ f_j^k &= \int_{x_i}^{x_{i+1}} f(x)v_j(x)dx \end{aligned} \tag{7}$$

By inserting the shape functions given above, $\mathbf{A}^k$ becomes:

$$\begin{aligned} \mathbf{A}^k &= \begin{bmatrix} \int_{x_i}^{x_{i+1}} \frac{-1}{h}\frac{-1}{h}dx & \int_{x_i}^{x_{i+1}} \frac{-1}{h}\frac{1}{h}dx \\ \int_{x_i}^{x_{i+1}} \frac{1}{h}\frac{-1}{h}dx & \int_{x_i}^{x_{i+1}} \frac{1}{h}\frac{1}{h}dx \end{bmatrix} \\ &= \begin{bmatrix} 1/h & -1/h \\ -1/h & 1/h \end{bmatrix} \end{aligned} \tag{8}$$

And the load vector becomes, with $f = 1$ chosen for simplicity:

$$\begin{aligned} \mathbf{f}^k &= \begin{bmatrix} \int_{x_i}^{x_{i+1}} \frac{x_{i+1}-x}{h}dx \\ \int_{x_i}^{x_{i+1}} \frac{x-x_i}{h}dx \end{bmatrix} \\ &= \begin{bmatrix} \frac{(x_{i+1}-x_i)^2}{2h} \\ \frac{(x_{i+1}-x_i)^2}{2h} \end{bmatrix} \\ &= \begin{bmatrix} h/2 \\ h/2 \end{bmatrix} \end{aligned} \tag{9}$$

Then, by the connectivity matrix relating how the nodes touch each other, the above matrices are "tesselated" along the diagonal of the global $\mathbf{A}$ and $\mathbf{f}$. Suppose for the time being that $u(1) = 0$, i.e. both ends have homogeneous Dirichlet conditions. Then, the matrix system before application of the Dirichlet conditions is:

$$\begin{bmatrix} 1/h & -1/h & & & \\ -1/h & 2/h & -1/h & & \\ & -1/h & 2/h & -1/h & \\ & & -1/h & 2/h & -1/h \\ & & & & \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n+1} \end{bmatrix} = \begin{bmatrix} h/2 \\ h \\ h \\ \vdots \\ h \\ h/2 \end{bmatrix} \tag{10}$$

Now, to apply the Neumann condition at $x = 1$, the load vector for the last element (the element spanning $1 - h \leq x \leq 1$ must be modified:

$$\begin{aligned} \mathbf{f}^{e=n} &= \begin{bmatrix} \int_{x_i}^{x_{i+1}} \frac{x_{i+1}-x}{h}dx \\ \int_{x_i}^{x_{i+1}} \frac{x-x_i}{h}dx + g \end{bmatrix} \\ &= \begin{bmatrix} \frac{(x_{i+1}-x_i)^2}{2h} \\ \frac{(x_{i+1}-x_i)^2}{2h} + g \end{bmatrix} \\ &= \begin{bmatrix} h/2 \\ h/2 + g \end{bmatrix} \end{aligned} \tag{11}$$

Then, the last equation in Eq. (13) must be modified. To strongly apply the homogeneous Dirichlet condition at $x = 0$, the first row and column of $\mathbf{A}$ is eliminated, and post-processing will assign the value $u_1 = 0$. The full matrix system, with boundary conditions $u(0) = 0, u'(1) = g$, is, for a four-element system (purely for illustration):

$$\begin{bmatrix} 2/h & -1/h & 0 & 0 \\ -1/h & 2/h & -1/h & 0 \\ 0 & -1/h & 2/h & -1/h \\ 0 & 0 & -1/h & 1/h \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} h \\ h \\ h \\ h/2+g \end{bmatrix} \tag{12}$$

Divide both sides by h for future insight into our choice of $f(x) = 1$:

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ (h/2+g)/h \end{bmatrix} \tag{13}$$

The central difference approximation to a second derivative is:

$$u''(x) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \tag{14}$$

Looking at the equation for node 3, for instance, we can express this equation for $i = 3$ in terms of the values on either side ($u_2 = u_{i-1}$ and $u_4 = u_{i+1}$):

$$-\frac{1}{h^2}\left(u_{i-1} - 2u_i + u_{i+1}\right) = 1 \tag{15}$$

Eq. (15) is identical to central difference approximation of the original equation in Eq. (1) - the second derivative is approximated as a central difference, while the right-hand-side reflects the (arbitrary) choice of setting $f(x) = 1$. The above equation holds for all nodes on the interior. The enforcement of the Neumann condition can also be seen analogous to a finite difference approximation. The equation for the node on the Neumann boundary (node 5, where $i = 5$), is:

$$\frac{1}{h}(-u_{i-1} + u_i) = \frac{h}{2} + g \tag{16}$$

It is clear to see that the above represents a forward Euler method, since the slope at node i depends only on the solution value at $i - 1$. The $h/2$ appears from the tessellation of the element load vector in $\mathbf{f}$, and hence if that part is neglected for the time begin, then the above is exactly equivalent to stating that:

$$u'(1) \approx \frac{u_5 - u_4}{h} = g + \frac{h}{2} f(x_{i-1}) \tag{17}$$

where the additional term on the right-hand side has been added as a correction term that results in a second-order approximation to the first derivative. So, this shows that, for linear Lagrange elements in 1-D, the finite element method is equivalent to a central difference method on the interior nodes with a second order finite difference method enforcing Neumann conditions on the boundaries through the use of a correction factor. So, for this simple case, the finite element method is equivalent to a second order finite difference approximation of the governing equation.

2

This question will solve the following boundary value problem on $x \in [0, 1]$:

$$u''''(x) = f(x) = 480x - 120 \tag{18}$$

with boundary conditions $u(0) = u'(0) = u(1) = u'(1) = 0$.

2.1

The Galerkin formulation seeks a solution u_h in the function space V_h, which is the same function space as the weight function v:

$$V_h = \{v \in C^1([0,1]) : v|_K \in P_3(K) \forall K \in T_h, v(0) = v(1) = 0 = v'(0) = v'(1)\} \tag{19}$$

The only requirement on the function space V_h is that it satisfy the homogeneous form of the essential boundary conditions. For a fourth-order governing equation, the essential boundary conditions are represented by the values of u and u' on the boundaries, while the natural boundary conditions in this case are the values of u''' and u'' on the boundaries. In order for v to be able to satisfy the essential boundary conditions, v must be of sufficiently high order to specify v and v' on the boundaries (which for a 1-D element requires four degrees of freedom per element, neglecting continuity and boundary conditions requirements for the time being). Multiplying the governing equation by a weight function $v(x)$ and integrating over the domain gives the weighted residual formulation of the governing equation:

$$\int_0^1 u''''(x)v(x)dx = \int_0^1 f(x)v(x)dx \tag{20}$$

Integrating by parts two times:

$$\begin{aligned} -\int_0^1 u'''(x)v'(x)dx + \left[u'''(x)v(x)\right]_0^1 &= \int_0^1 f(x)v(x)dx \\ \int_0^1 u''(x)v''(x)dx + \left[u'''(x)v(x)\right]_0^1 - \left[u''(x)v'(x)\right]_0^1 &= \int_0^1 f(x)v(x)dx \end{aligned} \tag{21}$$

Then, applying the boundary conditions:

$$\int_0^1 u''(x)v''(x)dx = \int_0^1 f(x)v(x)dx \quad \forall v \in V_h \tag{22}$$

Now that differentiation has been modified to act equally (to the extent possible based on the governing equation) on u and v, the above represents the weak form. The above equation holds for both the true solution u and the finite element solution u_h, and hence the above could also be written as follows, which matches that shown in this homework assignment:

$$\int_0^1 u_h''(x)v''(x)dx = \int_0^1 f(x)v(x)dx \quad \forall v \in V_h \tag{23}$$

2.2

Over a particular element, the solution is specified as a summation of coefficients a multiplied by basis functions φ, where N is the number of shape functions per element (which for linear Lagrange elements is equivalent to the number of nodes per element):

$$u_h^e(x) = \sum_{i=1}^{N} a_i \varphi_i(x) \tag{24}$$

The above expression uses an e in the superscript of u_h to refer to the fact that u_h^e is the solution only over a particular element e - we must find the solution over every element. Because the space is defined to contain cubic polynomials over each element, since we require four degrees of freedom per element (not yet assuming continuity between elements), the shape functions for this problem must be functions that can specify both types of essential boundary conditions - the value and the first derivative on the two boundary nodes for each element. This requires four shape functions per element. Hermite shape functions allow specification of both the value and derivative at edge nodes. There are four shape functions per element. These shape functions are defined over an element defined on $[a, b]$ as:

$$
\begin{aligned}
H_1(a) = 1, H_1(b) = 0, \frac{dH_1(a)}{dx} = 0, \frac{dH_1(b)}{dx} = 0 \\
H_2(a) = 0, H_2(b) = 1, \frac{dH_2(a)}{dx} = 0, \frac{dH_2(b)}{dx} = 0 \\
H_3(a) = 0, H_3(b) = 0, \frac{dH_3(a)}{dx} = 1, \frac{dH_3(b)}{dx} = 0 \\
H_4(a) = 0, H_4(b) = 0, \frac{dH_4(a)}{dx} = 0, \frac{dH_4(b)}{dx} = 1
\end{aligned} \tag{25}
$$

where H is used to represent the fact that these shape functions ψ are Hermite shape functions. For each of these shape functions, because they are cubic, they are uniquely represented by four coefficients c, d, e, f for a polynomial of the form:

$$
\begin{aligned}
H_1(x) =& c_1x^3 + d_1x^2 + e_1x + f_1 \\
H_2(x) =& c_2x^3 + d_2x^2 + e_2x + f_2 \\
H_3(x) =& c_3x^3 + d_3x^2 + e_3x + f_3 \\
H_4(x) =& c_4x^3 + d_4x^2 + e_4x + f_4
\end{aligned} \tag{26}
$$

$$
\begin{aligned}
\frac{dH_1(x)}{dx} =& 3c_1x^2 + 2d_1x + e_1 \\
\frac{dH_2(x)}{dx} =& 3c_2x^2 + 2d_2x + e_2 \\
\frac{dH_3(x)}{dx} =& 3c_3x^2 + 2d_3x + e_3 \\
\frac{dH_4(x)}{dx} =& 3c_4x^2 + 2d_4x + e_4
\end{aligned} \tag{27}
$$

So, for each element in the triangulation, these 16 coefficients must be determined by solving the following linear system for *each* of the functions H_i:

$$
\begin{bmatrix} a^3 & a^2 & a & 1 \\ b^3 & b^2 & b & 1 \\ 3a^2 & 2a & 1 & 0 \\ 3b^2 & 2b & 1 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ d_1 \\ e_1 \\ f_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{for } H_1 \tag{28}
$$

$$
\begin{bmatrix} a^3 & a^2 & a & 1 \\ b^3 & b^2 & b & 1 \\ 3a^2 & 2a & 1 & 0 \\ 3b^2 & 2b & 1 & 0 \end{bmatrix} \begin{bmatrix} c_2 \\ d_2 \\ e_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{for } H_2 \tag{29}
$$

$$
\begin{bmatrix} a^3 & a^2 & a & 1 \\ b^3 & b^2 & b & 1 \\ 3a^2 & 2a & 1 & 0 \\ 3b^2 & 2b & 1 & 0 \end{bmatrix} \begin{bmatrix} c_3 \\ d_3 \\ e_3 \\ f_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{for } H_3 \tag{30}
$$

$$
\begin{bmatrix} a^3 & a^2 & a & 1 \\ b^3 & b^2 & b & 1 \\ 3a^2 & 2a & 1 & 0 \\ 3b^2 & 2b & 1 & 0 \end{bmatrix} \begin{bmatrix} c_4 \\ d_4 \\ e_4 \\ f_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix} \quad \text{for } H_4 \tag{31}
$$

Then, this process is repeated for all the elements in the triangulation, where a and b will change corresponding to the endpoints of the domain. This system of equations is solved in Python, where the code is provided in the Appendix. This solution, for the two-element triangulation given, for superscripts that indicate the element number, is:

$$
\begin{aligned}
H_1^1(x) =& 16x^3 - 12x^2 + 1 \\
H_2^1(x) =& -16x^3 + 12x^2 \\
H_3^1(x) =& 4x^3 - 4x^2 + x \\
H_4^1(x) =& 4x^3 - 2x^2
\end{aligned}
\tag{32}
$$

$$
\begin{aligned}
H_1^2(x) =& 16x^3 - 36x^2 + 24x - 4 \\
H_2^2(x) =& -16x^3 + 36x^2 - 24x + 5 \\
H_3^2(x) =& 4x^3 - 10x^2 + 8x - 2 \\
H_4^2(x) =& 4x^3 - 8x^2 + 5x - 1
\end{aligned}
\tag{33}
$$

The second derivatives of these functions are needed for the finite element formulation, so they are computed as:

$$
\begin{aligned}
\frac{d^2 H_1^1(x)}{dx^2} =& 96x - 24 \\
\frac{d^2 H_2^1(x)}{dx^2} =& -96x + 24 \\
\frac{d^2 H_3^1(x)}{dx^2} =& 24x - 8 \\
\frac{d^2 H_4^1(x)}{dx^2} =& 24x - 4
\end{aligned}
\tag{34}
$$

$$
\begin{aligned}
\frac{d^2 H_1^2(x)}{dx^2} =& 96x - 72 \\
\frac{d^2 H_2^2(x)}{dx^2} =& -96x + 72 \\
\frac{d^2 H_3^2(x)}{dx^2} =& 24 - 20 \\
\frac{d^2 H_4^2(x)}{dx^2} =& 24 - 16
\end{aligned}
\tag{35}
$$

Fig. 1 shows the shape functions over the two-element domain. The basis over the domain is therefore these eight Hermite functions. These functions correspond to eight degrees of freedom, which are reduced to two when considering that the values and first derivatives at the single shared node must be continuous. Then, the four boundary conditions specify four of the degrees of freedom, leaving two degrees of freedom that are to be solved for using the finite element method. Because there are two degrees of freedom, the solution can be uniquely expressed as the sum of only two of these eight Hermite functions (this is specific to this particular problem).

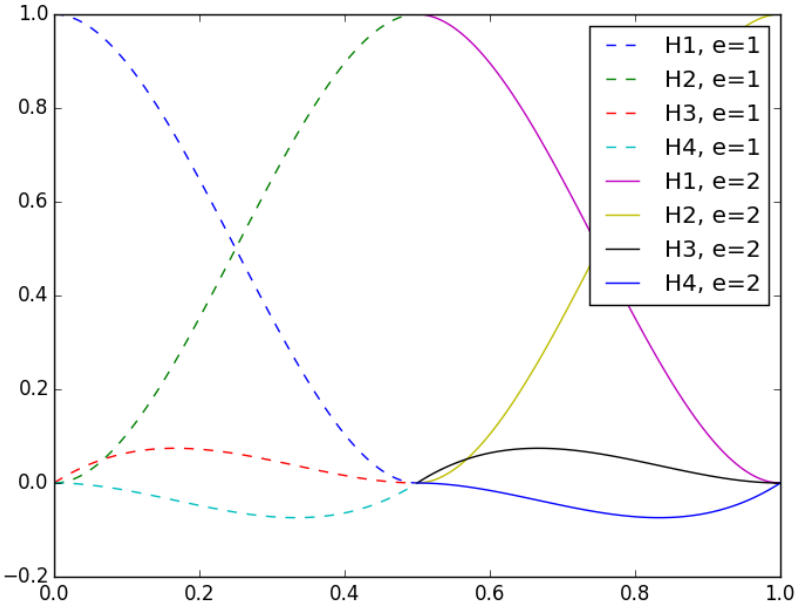


Figure 1. Hermite shape functions over the two-element domain.

2.3

The elemental stiffness matrix is given by:

$$A_{ij}^k = \int_0^1 \frac{d^2 H_i^k(x)}{dx^2} \frac{d^2 H_j^k(x)}{dx^2} dx \tag{36}$$

The elemental load vector is given by:

$$f_j = \int_0^1 f(x) H_j^k(x) dx \tag{37}$$

For element 1:

$$\mathbf{A}^1 = \begin{bmatrix} \int_0^{0.5} \frac{d^2 H_1^1(x)}{dx^2} \frac{d^2 H_1^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_1^1(x)}{dx^2} \frac{d^2 H_2^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_1^1(x)}{dx^2} \frac{d^2 H_3^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_1^1(x)}{dx^2} \frac{d^2 H_4^1(x)}{dx^2} dx \\ \int_0^{0.5} \frac{d^2 H_2^1(x)}{dx^2} \frac{d^2 H_1^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_2^1(x)}{dx^2} \frac{d^2 H_2^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_2^1(x)}{dx^2} \frac{d^2 H_3^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_2^1(x)}{dx^2} \frac{d^2 H_4^1(x)}{dx^2} dx \\ \int_0^{0.5} \frac{d^2 H_3^1(x)}{dx^2} \frac{d^2 H_1^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_3^1(x)}{dx^2} \frac{d^2 H_2^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_3^1(x)}{dx^2} \frac{d^2 H_3^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_3^1(x)}{dx^2} \frac{d^2 H_4^1(x)}{dx^2} dx \\ \int_0^{0.5} \frac{d^2 H_4^1(x)}{dx^2} \frac{d^2 H_1^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_4^1(x)}{dx^2} \frac{d^2 H_2^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_4^1(x)}{dx^2} \frac{d^2 H_3^1(x)}{dx^2} dx & \int_0^{0.5} \frac{d^2 H_4^1(x)}{dx^2} \frac{d^2 H_4^1(x)}{dx^2} dx \end{bmatrix} \tag{38}$$

The local stiffness matrix is computed in the same method for element 2, except that the integration bounds change from $[0, 0.5]$ to $[0.5, 1.0]$. It should be noted that the elemental basis functions are only nonzero over the particular element, so $\mathbf{A}^k$ is the same for both elements 1 and 2. Performing the computation shown above, the elemental stiffness matrices are (where the numbering given previously for the Hermite shape functions is changed so that 1 and 2 correspond to functions that obtain values and derivatives of unity at the left side, whereas 3 and 4 are reserved for the right side):

$$\mathbf{A}^1 = \mathbf{A}^2 = \begin{bmatrix} 96 & 24 & -96 & 24 \\ 24 & 8 & -24 & 4 \\ -96 & -24 & 96 & -24 \\ 24 & 4 & -24 & 8 \end{bmatrix} \tag{39}$$

And the elemental load vectors are:

$$\begin{aligned}\mathbf{f}^{\,1} &= \begin{bmatrix}-12 & -0.5 & 12 & -0.5\end{bmatrix}^T \\ \mathbf{f}^{\,2} &= \begin{bmatrix}48 & 4.5 & 72 & -5.5\end{bmatrix}^T\end{aligned} \tag{40}$$

And the global system to be solved is:

$$\begin{bmatrix} 96 & 24 & -96 & 24 & 0 & 0 \\ 24 & 8 & -24 & 4 & 0 & 0 \\ -96 & -24 & 96+96 & -24+24 & -96 & 24 \\ 24 & 4 & -24+24 & 8+8 & -24 & 4 \\ 0 & 0 & -96 & -24 & 96 & -24 \\ 0 & 0 & 24 & 4 & -24 & 8 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} -12 \\ -0.5 \\ 12+48 \\ -0.5+4.5 \\ 72 \\ -5.5 \end{bmatrix} \tag{41}$$

This is not the final system that is solved, however, since the essential boundary conditions must be taken into account in this formulation. The matrix system to be solved can be broken up into:

$$\begin{bmatrix} A_{uu} & A_{uk} \\ A_{ku} & A_{kk} \end{bmatrix} \begin{bmatrix} u_u \\ u_k \end{bmatrix} = \begin{bmatrix} f_u \\ f_k \end{bmatrix} \tag{42}$$

where the u and k subscripts indicate unknown (non-essential boundary condition locations) and known (essential boundary condition locations) values. So, instead of solving $\mathbf{Au} = \mathbf{f}$ using the full matrices above, the following system must be solved:

$$A_{uu}u_u = f_u - A_{uk}x_k \tag{43}$$

where $x_k = \mathbf{0}$ because all of the essential boundary conditions are homogeneous. From the boundary conditions, $u_1 = u_2 = u_5 = u_6 = 0$. So, the actual matrix system to be solved is:

$$\begin{bmatrix} 192 & 0 \\ 0 & 16 \end{bmatrix} \begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 60 \\ 4 \end{bmatrix} \tag{44}$$

Solving this system gives $u_3 = 5/16, u_4 = 1/4$, which then specifies the solution over the two elements according to Eq. (24):

$$\begin{aligned}u_h^1(x) =& u_3(-16x^3 - 12x^2 + 1) + u_4 * 4x^3 - 2x^2 \\ u_h^2(x) =& u_3(16x^3 - 36x^2 + 24x - 4) + u_4(4x^3 - 10x^2 + 8x - 2)\end{aligned} \tag{45}$$

The analytical solution to the governing equation is given by:

$$\begin{aligned}u(x) =& C_4x^3 + C_3x^2 + C_2x + C_1 + 4x^5 - 5x^4 \\ u'(x) =& 3C_4x^2 + 2C_3x + C_2 + 20x^4 - 20x^3\end{aligned} \tag{46}$$

Applying the boundary conditions to determine the unknown coefficients gives $C_1 = 0, C_2 = 0, C_3 = 3, C_4 = -2$, so the analytical solution is:

$$u(x) = -2x^3 + 3x^2 + 4x^5 - 5x^4 \tag{47}$$

Fig. 2 shows a comparison between the analytical solution above and the finite element solution. As can be seen, the finite element solution is a fairly good approximation to the true solution, and had there been more elements, would have much better approximated the true solution. The code developed for this section is included in the Appendix.

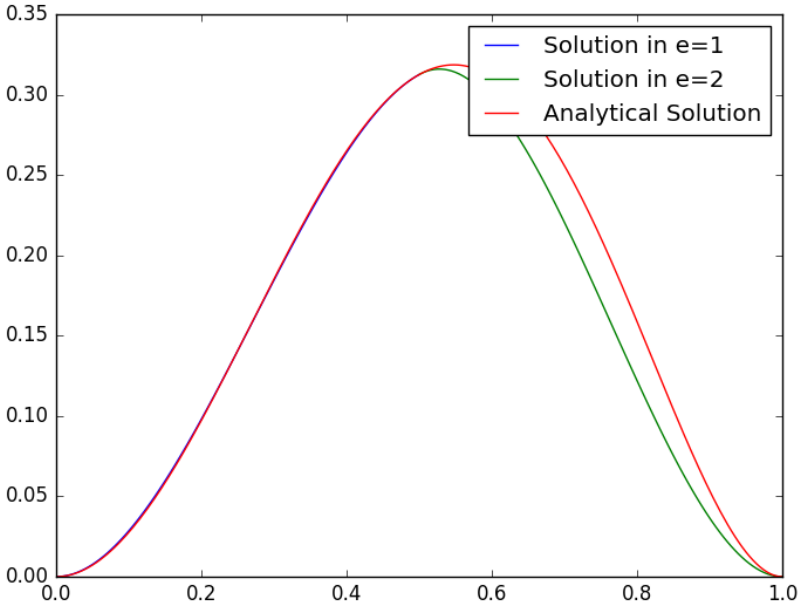


Figure 2. Comparison between the analytical and finite element solutions for the given problem statement.

3

This question solves the Poisson equation on an unstructured, triangular grid, subject to homogeneous Dirichlet and Neumann boundary conditions:

$$\begin{aligned} -\nabla^2 u(x,y) &= 1 \\ u|_{\Gamma_d} = 0, \quad \hat{n} \cdot \nabla u|_{\Gamma_t} &= 0 \end{aligned} \tag{48}$$

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:

$$-\int_\Omega \nabla^2 u(x,y) v(x,y) d\Omega = \int_\Omega v(x,y) d\Omega \tag{49}$$

The weak form is obtained by integrating by parts:

$$\int_\Omega \nabla u(x,y) \cdot \nabla v(x,y) d\Omega - \int_\Gamma \hat{n} \cdot \nabla u(x,y) v(x,y) d\Gamma = \int_\Omega v(x,y) d\Omega \tag{50}$$

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.

$$\int_\Omega \nabla u(x,y) \cdot \nabla v(x,y) d\Omega = \int_\Omega v(x,y) d\Omega \tag{51}$$

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

$$u_h = \sum_{i=1}^{N} a_j \psi_j(x,y) \tag{52}$$

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.

$$u_h^k = \sum_{i=1}^{n_{en}} a_j \psi_j(x,y) = \mathbf{N}\mathbf{u} \tag{53}$$

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

$$\mathbf{N} = \begin{bmatrix} \psi_1(x,y) & \psi_2(x,y) & \cdots & \psi_{n_{en}} \end{bmatrix} \tag{54}$$

$$\mathbf{u} = \begin{bmatrix} a_1 & a_2 & \cdots & a_{n_{en}} \end{bmatrix}^T \tag{55}$$

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:

$$\nabla u_h = \frac{\partial u_h}{\partial x}\hat{x} + \frac{\partial u_h}{\partial y}\hat{y} = \mathbf{B}\mathbf{u} \tag{56}$$

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

$$\mathbf{B} = \begin{bmatrix} \frac{\partial \psi_1}{dx} & \frac{\partial \psi_2}{dx} & \cdots & \frac{\partial \psi_{n_e n}}{dx} \\ \frac{\partial \psi_1}{dy} & \frac{\partial \psi_2}{dy} & \cdots & \frac{\partial \psi_{n_e n}}{dy} \end{bmatrix} \tag{57}$$

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:

$$\begin{aligned} \int_\Omega (\mathbf{B}\mathbf{u})^T(\mathbf{B}\mathbf{v})d\Omega &= \int_\Omega \mathbf{N}\mathbf{v}d\Omega \\ \int_\Omega \mathbf{B}^T\mathbf{u}^T(\mathbf{B}\mathbf{v})d\Omega &= \int_\Omega \mathbf{N}\mathbf{v}d\Omega \\ \int_\Omega \mathbf{B}^T\mathbf{u}\mathbf{B}d\Omega &= \int_\Omega \mathbf{N}d\Omega \end{aligned} \tag{58}$$

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:

$$\mathbf{A}^k = \int_{\Omega_k} \mathbf{B}^T\mathbf{B}d\Omega \tag{59}$$

$$\mathbf{F}^k = \int_{\Omega_k} \mathbf{N}d\Omega \tag{60}$$

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. If these shape functions are determined individually for each element, they would be determined by solving the following

system for the three different right-hand sides shown below to give the coefficients a, b, c for each of the three shape functions (assuming triangular elements for this problem):

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} ; \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} ; \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{61}$$

where x_i, y_i is the i coordinate of the triangle, for $i = 1, 2, 3$. 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 $(0,0), (1,0), (0,1)$ 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 $(x_1 = 0, y_1 = 0, (x_2 = 1, y_2 = 0), (x_3 = 0, y_3 = 1)$, the shape functions in the master domain are:

$$\begin{aligned} \psi_1(\xi, \eta) =& 1 - \xi - \eta \\ \psi_2(\xi, \eta) =& \xi \\ \psi_3(\xi, \eta) =& \eta \end{aligned} \tag{62}$$

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)

$$\begin{aligned} x(\xi, \eta) =& \sum_{i=1}^{n_{en}} X_i \psi_i(\xi, \eta) \\ y(\xi, \eta) =& \sum_{i=1}^{n_{en}} Y_i \psi_i(\xi, \eta) \end{aligned} \tag{63}$$

To transform the integrals appearing in Eq. (66) and (67) 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:

$$\begin{aligned} d\vec{x} =& \mathbf{D} d\vec{\xi} \\ \begin{bmatrix} dx \\ dy \end{bmatrix} =& \begin{bmatrix} dx/d\xi & dx/d\eta \\ dy/d\xi & dy/d\eta \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \end{aligned} \tag{64}$$

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:

$$\begin{aligned} \frac{dx(\xi, \eta)}{d\xi} =& \sum_{i=1}^{n_{en}} X_i \frac{d\psi_i(\xi, \eta)}{d\xi} \\ \frac{dx(\xi, \eta)}{d\eta} =& \sum_{i=1}^{n_{en}} X_i \frac{d\psi_i(\xi, \eta)}{d\eta} \\ \frac{dy(\xi, \eta)}{d\xi} =& \sum_{i=1}^{n_{en}} Y_i \frac{d\psi_i(\xi, \eta)}{d\xi} \\ \frac{dy(\xi, \eta)}{d\eta} =& \sum_{i=1}^{n_{en}} Y_i \frac{d\psi_i(\xi, \eta)}{d\eta} \end{aligned} \tag{65}$$

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

$$\mathbf{A}^k = \int_0^1 \int_0^{1-\xi} (\mathbf{D}^{-1}\mathbf{C})^T (\mathbf{D}^{-1}\mathbf{C}) |\mathbf{D}| d\eta d\xi \tag{66}$$

$$\mathbf{F}^k = \int_0^1 \int_0^{1-\xi} \mathbf{N} |\mathbf{D}| d\eta d\xi \tag{67}$$

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:

$$\mathbf{C} = \begin{bmatrix} \frac{\partial \psi_1}{d\xi} & \frac{\partial \psi_2}{d\xi} & \dots & \frac{\partial \psi_{n_e n}}{d\xi} \\ \frac{\partial \psi_1}{d\eta} & \frac{\partial \psi_2}{d\eta} & \dots & \frac{\partial \psi_{n_e n}}{d\eta} \end{bmatrix} \tag{68}$$

Note that the integrals in Eq. (66) and (67) 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:

$$\int_0^1 \int_0^{1-\xi} f(\xi,\eta) d\eta d\xi \approx \sum_{i=1}^{n_{qp}} w_i f(\xi_i, \eta_i) \tag{69}$$

For $n_{qp} = 1$, the following one-point rule exactly integrates linear integrands:

$$\begin{aligned} w =& [1/2] \\ (\xi_i, \eta_i) =& [1/3, 1/3] \end{aligned} \tag{70}$$

For $n_{qp} = 3$, the following one-point rule exactly integrates quadratic integrands:

$$\begin{aligned} w =& [1/6; 1/6; 1/6] \\ (\xi_i, \eta_i) =& [1/6, 1/6; 2/3, 1/6; 1/6, 2/3] \end{aligned} \tag{71}$$

Because the highest order expected in the integrands for this problem occurs in $\mathbf{F}$, which has an integrand that is at most order 1, the one-point quadrature rule is used. 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:

$$\mathbf{LM} = \begin{bmatrix} 1 & 2 & 5 & 4 \\ 2 & 3 & 6 & 5 \\ 4 & 5 & 8 & 7 \\ 5 & 6 & 9 & 8 \end{bmatrix} \tag{72}$$

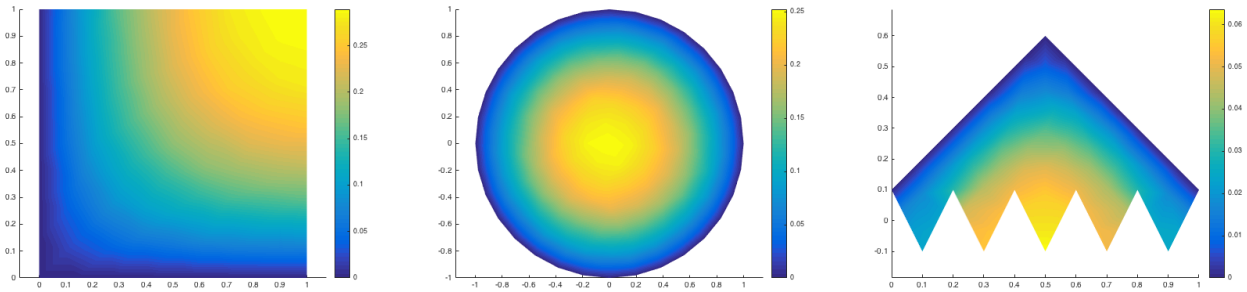
where the local nodes are numbered in a counterclockwise manner beginning from the bottom left node. Then, for example, the second row in the global stiffness matrix would be assembled as:

$$\mathbf{K}(2,:) = \begin{bmatrix} k_{2,1}^{e=1}, & k_{2,2}^{e=1} + k_{1,1}^{e=2}, & k_{1,2}^{e=2}, & k_{2,4}^{e=1}, & k_{1,4}^{e=2} + k_{2,3}^{e=1}, & k_{1,3}^{e=2}, & 0, & 0, & 0 \end{bmatrix} \tag{73}$$

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:

$$A_{uu} u_u = f_u - A_{uk} x_k \tag{74}$$

where $x_k = \mathbf{0}$ because all of the essential boundary conditions are homogeneous. The approach just described is, in general, how finite element codes are written. However, for this assignment, because the problem is relatively simple, the integrals over the triangles can be computed directly based on knowledge of the corner nodes. Hence, the integrals are actually hard-coded into the code submitted. Fig. 3 shows the solutions obtained for the three test cases provided in the assignment.



(a) Case 1 **(b)** Case 2 **(c)** Case 3

Figure 3. Finite element solutions u_h for the three geometries provided in the assignment.

In addition, Fig. 7 shows the solution for a user-defined polygon.

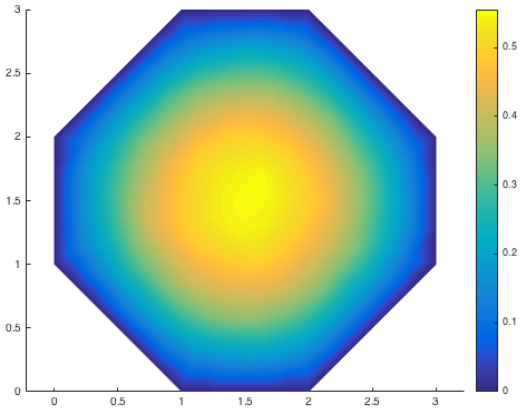


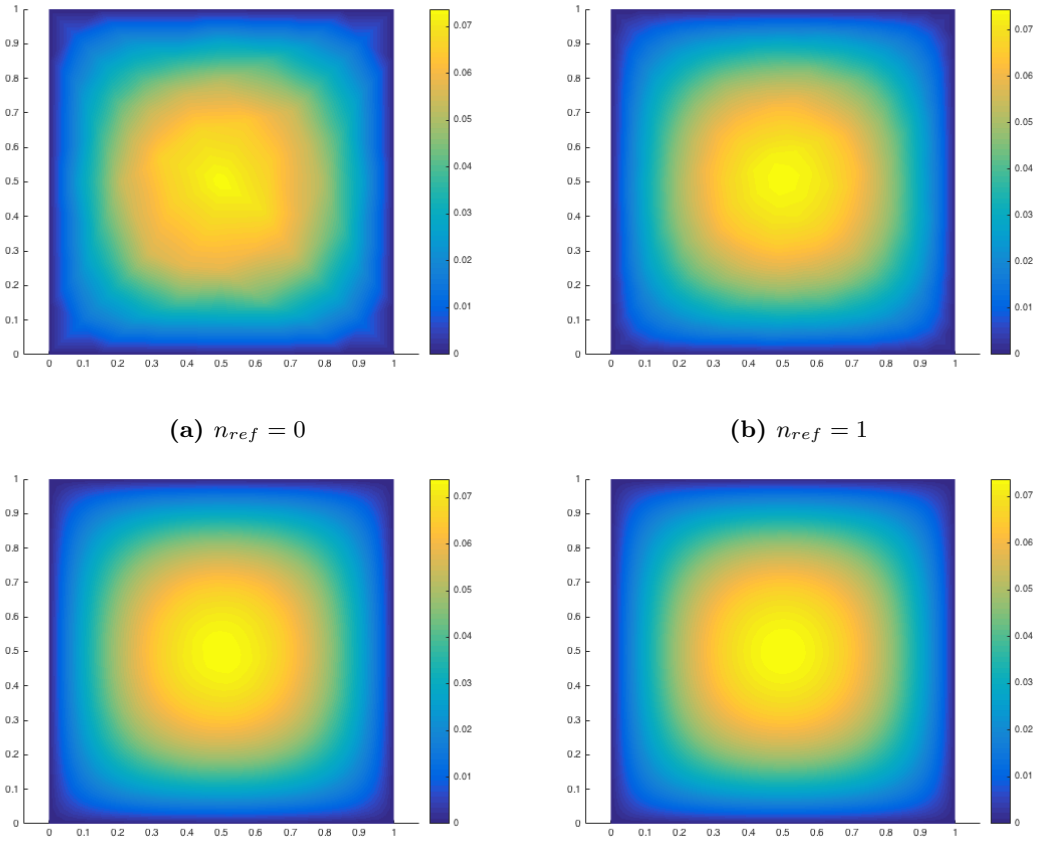
Figure 4. Finite element solution u_h for a fourth geometry.

All of the code developed for this section is shown in the Appendix. Not shown are the meshing files developed from the previous homework, since they are largely the same and are still included in the zip file.

4

This problem investigates the convergence properties of the finite element method for linear triangular elements. By solving the same problem with a finer and finer mesh, with the exact solution assumed approximated very well by the finest mesh, the error of each solution with respect to the true solution can be estimated as a function of the mesh refinement. The finer the mesh, the more accurate the solution (as expected due to the higher number of degrees of freedom to capture the behavior), and this investigation will show just how much more accurate the solution is for each successive mesh refinement. This is important knowledge if the tradeoff between more accurate results and longer computation times is a significant factor.

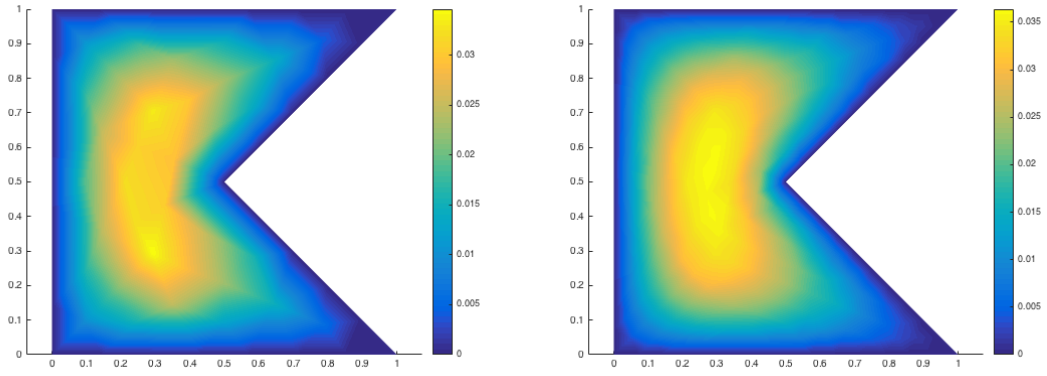
Fig. 5 and 6 shows the solutions obtained for various levels of refinement for the two problem domains specified.



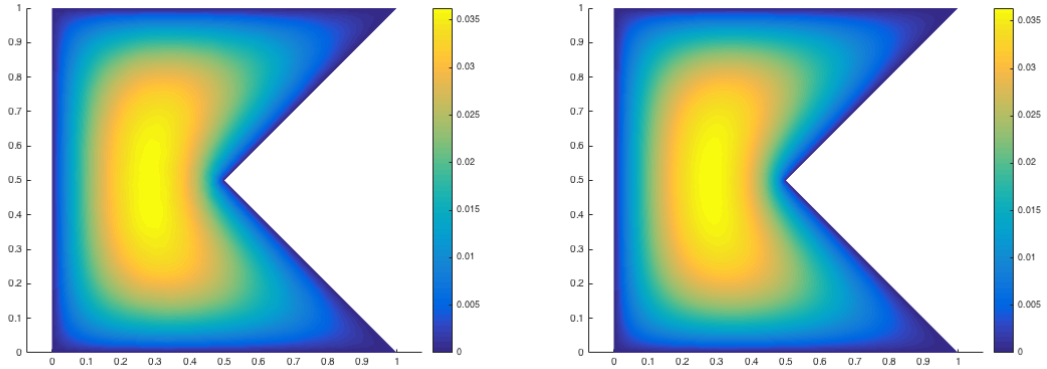
(a) $n_{ref} = 0$ **(b)** $n_{ref} = 1$

(c) $n_{ref} = 2$ **(d)** $n_{ref} = 3$

Figure 5. Finite element solutions u_h for the square geometry provided in the assignment.



(a) $n_{ref} = 0$ **(b)** $n_{ref} = 1$



(c) $n_{ref} = 2$ **(d)** $n_{ref} = 3$

Figure 6. Finite element solutions u_h for the polygon geometry provided in the assignment.

The error in each solution is computed as the maximum norm of the errors at the nodes - a loop circulates through all the nodes of the most refined mesh and determines which nodes are shared by the coarser meshes. The maximum norm is defined as the maximum of the absolute difference between the solution values at all the nodes. The slope of these lines on the log-log coordinates is 2.0267 for the square domain and 1.1097 for the polygon domain. This can be interpreted in the following manner - by halving the mesh spacing (as is achieved through a single uniform refinement), the error decreases by a factor of about 2 for the square domain and 1 for the polygonal domain. For linear elements, the finite element method in most situations can be interpreted as being equivalent to a second order finite difference scheme, and so it is to be expected that a convergence rate of 2 be obtained with the square domain. However, some other effect is distorting this convergence rate for the polygonal domain (which we are to talk about soon in lecture, the professor says!).

5 Appendix

5.1 Code, Question 2

This Python script was used to determine the appropriate Hermite polynomials and the elemental stiffness matrix and elemental load vectors. Because the matrix system reduced to a simple 2×2 system, the actual solution was performed by hand.

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

```
import sympy as sp

# number of elements
num_elem = 2;

# coordinates of the nodes
p = np.array([[0.0, 0.5], [0.5, 1.0]])

for j in range(num_elem):
                    for i in range(4):
                                        a = p[j, 0]
                                        b = p[j, 1]
                                        A = np.array([[a**3, a**2, a, 1], [b**3, b**2,
                                            ↪ b, 1], \
                                                            [3*(a**2), 2*a, 1, 0], [3*(b
                                                                ↪ **2), 2*b, 1, 0]])

                                        B = np.empty([4, 1])
                                        B[i] = 1
                                        x = np.linalg.solve(A, B)

# plot the shape functions
xx = np.linspace(0.0, 0.5, 100)
xxx = np.linspace(0.5, 1.0, 100)
domain = np.linspace(0.0, 1.0, 200)

H1_1 = 16*(xx**3) - 12*(xx**2) + 1
H3_1 = -16*(xx**3) + 12*(xx**2)
H2_1 = 4*(xx**3) - 4*(xx**2) + xx
H4_1 = 4*(xx**3) - 2*(xx**2)

H1_2 = 16*(xxx**3) - 36*(xxx**2) + 24*xxx - 4
H3_2 = -16*(xxx**3) + 36*(xxx**2) - 24*xxx + 5
H2_2 = 4*(xxx**3) - 10*(xxx**2) + 8*xxx - 2
H4_2 = 4*(xxx**3) - 8*(xxx**2) + 5*xxx - 1

plt.plot(xx, (5.0/16.0)*H3_1 + 0.25*H4_1, label='Solution in e=1')
plt.plot(xxx, (5.0/16.0)*H1_2 + 0.25*H2_2, label='Solution in e=2')
plt.plot(domain, -2*(domain**3) + 3*(domain**2) + 4*(domain**5) - 5*(domain
    ↪ **4), label='Analytical Solution')
plt.legend()
plt.savefig('q3_soln.png')

#plt.plot(xx, H1_1, linestyle='--', label='H1, e=1')
#plt.plot(xx, H2_1, linestyle='--', label='H2, e=1')
#plt.plot(xx, H3_1, linestyle='--',label='H3, e=1')
#plt.plot(xx, H4_1, linestyle='--',label='H4, e=1')
#plt.plot(xxx, H1_2, label='H1, e=2')
#plt.plot(xxx, H2_2, label='H2, e=2')
#plt.plot(xxx, H3_2, label='H3, e=2')
#plt.plot(xxx, H4_2, label='H4, e=2')
#plt.legend()

#plt.savefig('q2_Hermite_functions.png')
```

```
# compute elemntal stiffness matrix K
x = sp.Symbol('x')

H1_1 = 16*(x**3) - 12*(x**2) + 1
H3_1 = -16*(x**3) + 12*(x**2)
H2_1 = 4*(x**3) - 4*(x**2) + x
H4_1 = 4*(x**3) - 2*(x**2)

H1_2 = 16*(x**3) - 36*(x**2) + 24*x - 4
H3_2 = -16*(x**3) + 36*(x**2) - 24*x + 5
H2_2 = 4*(x**3) - 10*(x**2) + 8*x - 2
H4_2 = 4*(x**3) - 8*(x**2) + 5*x - 1

k = np.empty([4, 4])

f = np.empty([4, 1])
for e in range(num_elem):
                for i in range(4):
                                f[i] = sp.integrate((480*x-120)*eval("H{0}_{1}
                                    ↪ ".format(i+1, e+1)), (x, p[e, 0], p[e,
                                    ↪ 1]))
                                for j in range(4):
                                                integrand = sp.diff(eval("H{0}
                                                    ↪ _{1}".format(i+1, e+1)),
                                                    ↪  x, 2) * sp.diff(eval("H
                                                    ↪ {0}_{1}".format(j+1, e
                                                    ↪ +1)), x, 2)
                                                k[i, j] = sp.integrate(
                                                    ↪ integrand, (x, p[e, 0],
                                                    ↪ p[e, 1]))
```

5.2 Code, Question 3

Not shown in this document are the meshing files that were submitted with the previous assignment. They are still submitted with this assignment, but there is no reason to include them again.

5.2.1 `fempoi.m`

This function implements the solution of the Poisson equation on an unstructured triangular mesh.

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

LM = t;
num_elem = length(LM(:,1));
dirichlet_nodes(1,:) = e;
num_nodes_per_elem = 3;          % linear triangular elements

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

```
% specify the boundary conditions - homogeneous neumann and dirichlet
dirichlet_nodes(2,:) = zeros .* dirichlet_nodes(1,:);
a_k = dirichlet_nodes(2,:);
num_nodes = length(p(:,1));

% 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;

    X = [p(LM(elem,1),1) p(LM(elem,2),1) p(LM(elem,3),1);
         p(LM(elem,1),2) p(LM(elem,2),2) p(LM(elem,3),2);
         1               1               1              ];
    C = inv(X);

    % compute area of triangle based on hard-coded formula
    area = polyarea([p(LM(elem,1),1) p(LM(elem,2),1) p(LM(elem,3),1)], [p(LM(
        ↪ elem,1),2) p(LM(elem,2),2) p(LM(elem,3),2)]);

    for l = 1:3
        f(l, 1) = area / 3;
        for ll = 1:3
            k(l, ll) = (C(ll, 1)*C(l, 1) + C(l, 2)*C(ll, 2)) * area;
        end
    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(LM(elem, i), LM(elem, j)) = K(LM(elem, i), LM(elem, j)) + k(i,j);
    end

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

% perform static condensation to remove known Dirichlet nodes from solve
[K_uu, K_uk, F_u, F_k] = condensation(K, F, num_nodes, dirichlet_nodes);

% perform the solve
a_u_condensed = K_uu \ (F_u - K_uk * dirichlet_nodes(2,:)');

% expand a_condensed to include the Dirichlet nodes
a = zeros(num_nodes, 1);

a_row = 1;
i = 1;      % index for dirichlet_nodes
```

```
j = 1;        % index for expanded row

for a_row = 1:num_nodes
    if (find(dirichlet_nodes(1, :) == a_row))
        a(a_row) = dirichlet_nodes(2,i);
        i = i + 1;
    else
        a(a_row) = a_u_condensed(j);
        j = j + 1;
    end
end

tplot(p, LM, a)

end
```

5.2.2 condensation.m

This function performs static condensation by removing the Dirichlet nodes from the solution so that the correct matrix system can be solved.

```
% Performs static condensation and removes Dirichlet nodes from the global
% matrix solve K * a = F

% To illustrate the process here, assume that the values at the first and
% last nodes (1 and 5) are specified. The other nodes (2, 3, and 4) are
% unknown. For a 5x5 node system, the following matrices are defined:

% K =           K(1,1)  K(1,2)  K(1,3)  K(1,4)  K(1,5)
%               K(2,1)  K(2,2)  K(2,3)  K(2,4)  K(2,5)
%               K(3,1)  K(3,2)  K(3,3)  K(3,4)  K(3,5)
%               K(4,1)  K(4,2)  K(4,3)  K(4,4)  K(4,5)
%               K(5,1)  K(5,2)  K(5,3)  K(5,4)  K(5,5)


% K_uu_rows =   K(2,1)  K(2,2)  K(2,3)  K(2,4)  K(2,5)
%               K(3,1)  K(3,2)  K(3,3)  K(3,4)  K(3,5)
%               K(4,1)  K(4,2)  K(4,3)  K(4,4)  K(4,5)



% K_uu =                K(2,2)  K(2,3)  K(2,4)
%                       K(3,2)  K(3,3)  K(3,4)
%                       K(4,2)  K(4,3)  K(4,4)



% K_uk =        K(2,1)                          K(2,5)
%               K(3,1)                          K(3,5)
%               K(4,1)                          K(4,5)



% K_ku =                K(1,2)  K(1,3)  K(1,4)
```

```
%
%
%
%                              K(5,2)   K(5,3)   K(5,4)



% K_kk =          K(1,1)                              K(1,5)
%
%
%
%                 K(5,1)                              K(5,5)

function [K_uu, K_uk, F_u, F_k] = condensation(K, F, num_nodes,
    ↪ dirichlet_nodes)

K_uu_rows = zeros(num_nodes - length(dirichlet_nodes(1, :)), num_nodes);
K_uk = zeros(num_nodes - length(dirichlet_nodes(1,:)), length(dirichlet_nodes
    ↪ (1,:)));
F_u = zeros(num_nodes - length(dirichlet_nodes(1, :)), 1);
F_k = zeros(length(dirichlet_nodes(1,:)), 1);

K_row = 1;
i = 1;        % index for dirichlet_nodes
j = 1;        % index for condensed row
l = 1;        % index for unknown condensed row
m = 1;        % index for known condensed row

for K_row = 1:num_nodes
    if (find(dirichlet_nodes(1, :) == K_row))
        F_k(m) = F(K_row);
        m = m + 1;
        i = i + 1;
    else
        K_uu_rows(j,:) = K(K_row,:);
        F_u(l) = F(K_row);
        j = j + 1;
        l = l + 1;

    end
end

% perform static condensation to remove Dirichlet node columns from solve
K_uu = zeros(num_nodes - length(dirichlet_nodes(1, :)), num_nodes - length(
    ↪ dirichlet_nodes(1, :)));

K_column = 1;
i = 1;            % index for dirichlet nodes
j = 1;            % index for condensed column
m = 1;            % index for K_uk column

for K_column = 1:num_nodes
    if (find(dirichlet_nodes(1, :) == K_column))
        K_uk(:,m) = K_uu_rows(:, K_column);
```

```
        m = m + 1;
        i = i + 1;
    else
        K_uu(:,j) = K_uu_rows(:, K_column);
        j = j + 1;
    end
end
```

5.2.3 permutation.m

This function permutes the numbers $i = 1, 2, \cdots, n_{en}$ to be used in filling in the global matrix and vector by the local element matrices and vectors.

```
function [permutation] = permutation(num_nodes_per_element)

permutation = zeros(num_nodes_per_element ^ 2, 2);

r = 1;
c = 1;
for i = 1:num_nodes_per_element^2
    permutation(i,:) = [r, c];
    if c == num_nodes_per_element
        c = 1;
        r = r + 1;
    else
        c = c + 1;
    end
end
```

5.3 Code, Question 4

5.3.1 poiconv.m

This function solves a finite element problem for multiple uniform refinement levels and estimates the error as the max-norm of the error at all of the shared nodes between the current refinement level and the maximum refinement level.

```
function [errors] = poiconv(pv, hmax, nrefmax)

errors = zeros(1, nrefmax);
tol = 1e-10;

% compute the reference solution
[p_ref, t, e] = pmesh(pv, hmax, nrefmax);
a_ref = fempoi(p_ref, t, e);

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

    k = 1;
    error = [];

    % loop over all the points in the ref soln to find the maximum error
```

```
    for i = 1:length(p_ref)
        % loop over all the points in the new soln
        for j = 1:length(p)
            % check if x-coordinates match
            if abs(p_ref(i, 1) - p(j, 1)) <= tol
                % check if y-coordinates match
                if abs(p_ref(i, 2) - p(j, 2)) <= tol
                    % compute the error at the shared node
                    error(k) = abs(a_ref(i) - a(j));
                    k = k + 1;
                end
            end
        end
    end

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

end
```