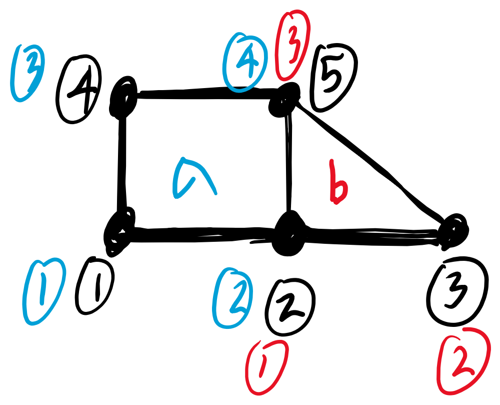
Homework 3

CESG 504 – Finite Element Methods

Due: Friday, May 7, 2021

1. ***Assembly:*** This problem will be our final venture into manual assembly of stiffness matrices. Hereafter, we will let computers do the assembly for us. It will focus on 2D problems, but if you can do it for 2D, you can do it for 3D (it is just way more tedious). Also, the idea works the same regardless of the type of problem/physics that are being solved. It is simply about mapping local to global.

   (a) A 2D heat transfer problem domain has been discretized into the two elements shown below (which would probably be way too coarse for most applications). For heat transfer (and flow through porous media, and torsion problems), there is one DOF per node, which is why the stiffness matrices have the dimensions that they do. Assemble the global stiffness matrix from the two local stiffness matrices. Show all components. Note: The local node numbers are shown in blue (a) and red (b), while the global numbering is in black. ***Instead of labeling the components with superscripts a and b, please use two colors as shown in here to make grading easier.***



$$[k^a] = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix}$$

$$[k^b] = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$$

Discussion 1: Note that at the heart of this method is the same piecewise integration concept we used in 1D. Piecewise integration would give the same overlapping at nodes that we see in this element assembly approach. If you have the time, please try to formalize how piecewise integration ends up still working the same way as it did for 1D problems.

Discussion 2: This problem clarifies how mixed element types can be made to work just fine (though continuity at the shared vertices is a potential concern depending on the types of shape function that are used).

(b) Assemble the global $\{P_{FEF}\}$ using the same color scheme.

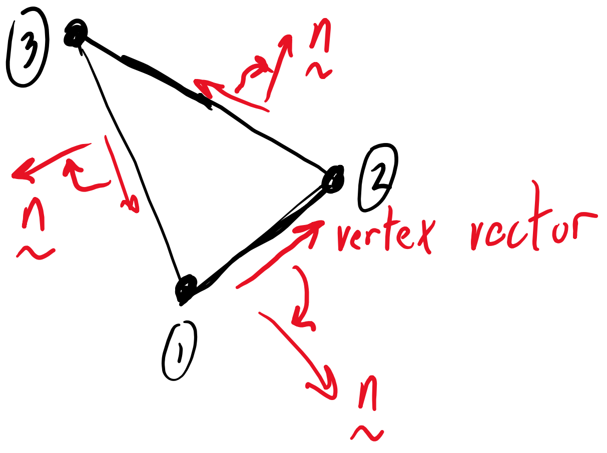
(c) What would the sizes of $[K^a]$, $[K^b]$, $[K]$ be this was instead a 2D elasticity problem?

(d) What are the physical meaning of the $\{P\}$ and $\{P_{FEF}\}$ vectors for the two kinds of physics applications listed below. For both cases, sketch the two-element model and place vectors in their correct location.

- *Heat conduction problems*
- *Flow through porous media*

(e) The last input in Discussion 4 on the next page is listed as Very Bad. Why might this input be even worse than the one above it which is only listed as Bad. Both of these inputs would result in clashes in most commercial FEM codes. The very bad one would produce some very weird errors that might look like trying to fold something inside out. (good thing the proper approach is usually automated within commercial software).

Discussion 3: It is common to list nodes counterclockwise (ccw) at the element level (at the global level it is not required), but why does this matter? This is a common convention, but it is not formally required, as long as the rest of the FEM code is consistent. The main reason for needing this convention is that it allows FEM code to consistently determine what is "outside" and what is "inside" the element. The value of knowing outside, is that the boundary integral terms (in the $\{P\}$ vector) are always dotted by an outward normal as opposed to an inward normal. If you know the elements are listed counter-clockwise, then the outward normal will can always be found by turning 90 degrees to the right of each vertex as shown below. Again, this would not be required as there could be other ways to find out what is the inside and what is outside.



i.e. Turn right 90° for outward normal

Discussion 4: How is this node numbering actually implemented? Somewhere in an FEM code, each element will be defined according to the global nodes it connects (note that this is actually a really interesting/difficult combinatorial math problem, especially in 3D). Typically, the nodes will be listed in ccw order. As an example, a few proper and improper ways to input the node numbers for element a of the two-element structure we started with are listed below (note that FEM packages automate all of this).

1,2,5,4

2,5,4,1

5,4,1,2

4,1,2,5

} All are acceptable ccw inputs for element a

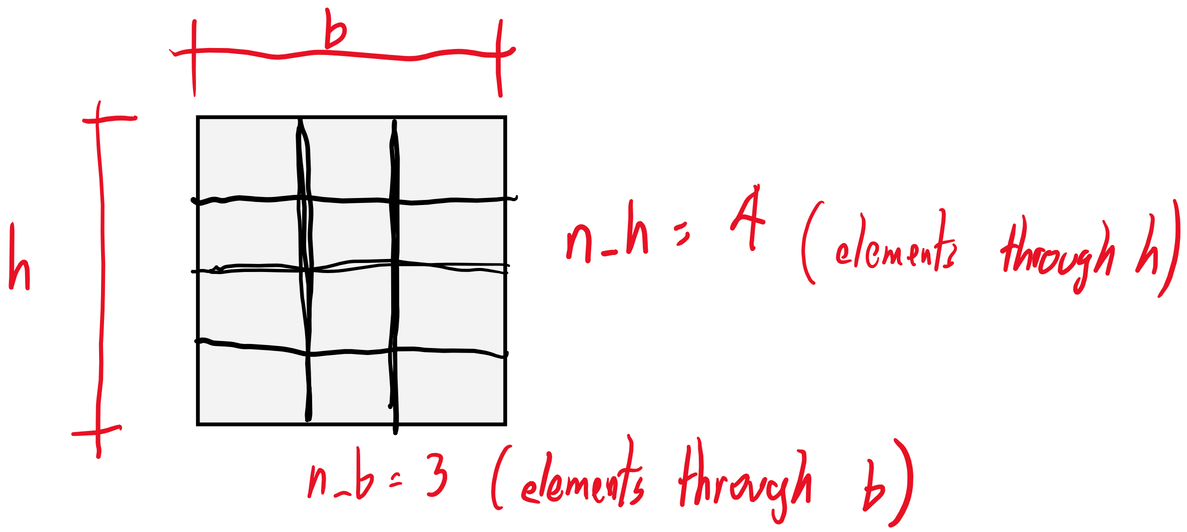
4,5,2,1 } example of improper CW input BAD

1,5,2,4 } very BAD, see (e)

2. ***1D/2D FEM Code:*** In this question we will create our own FEM code, using Matlab, to solve the Prandtl Stress Equation. With few modifications, the same code could be used for 2D heat conduction or flow through porous media. The code will use rectangular elements, and will work for rectangular cross-section. The different parts of the question will walk you through the important parts of the code.

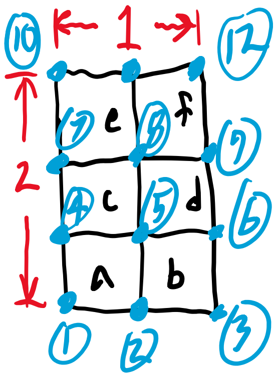
   Note: If you looked 'under the hood' of commercial FEM software, you would see the same stuff you will be coding here, though it may be in different programming languages. They have just had many years to add different elements, different physics regimes, and all sorts of GUIs/bells/whistles.

   (a) ***Inputs:*** The inputs at the very top of your code should be
   - 4 geometric properties b, h, n_b, n_h (see sketch below). Note that b and h don't need to be the same (i.e., this could be a rectangle), and the elements don't need to be square either, though the performance is better if the aspect ratio is not too large.
   - 1 Material property, G.
   - 1 Deformation property, theta.
   - Note, these things will be user inputs. It should work for any number of elements, not just for a 4x3 as shown.



(b) ***Mesher:*** This part should construct the nodal, **n_info**, and element, **e_info**, matrices from the inputs. An example is shown below, but your code should work for any values. The challenge here is to come up with algorithms to efficiently construct the two matrices for any user inputs. This part is all about bookkeeping. You can use any node numbering system you like (e.g., vertical strips as opposed to horizontal), but I recommend using the approach shown below. These two matrices will later be used to assembly the stiffness equations. **Hint:** It always helps to do a small hand problem on the side to test your algorithm.

$$\left.\begin{array}{l} b=1 \\ n\_b=2 \\ h=2 \\ n\_h=3 \end{array}\right\} \text{Inputs Representing} \rightarrow$$



$$n\_info = \begin{array}{cc|c} X\_coord & Y\_coord & \\ \hline 0 & 0 & 1 \\ 0.5 & 0 & 2 \\ 1 & 0 & 3 \\ 0 & 0.667 & 4 \\ \vdots & & \vdots \\ 1 & 2 & 12 \end{array}$$

i.e. Row # = Node #

$$e\_info = \begin{bmatrix} 1 & 2 & 5 & 4 \\ 2 & 3 & 6 & 5 \\ 4 & 5 & 8 & 7 \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{array}{c} a \\ b \\ c \\ \vdots \end{array}$$

i.e row # = element #

(c) ***Boundary Condition Definition:*** This part will construct the **id_f** and **id_s** vectors. These vectors denote the degrees of freedom (which are the same as the nodes when there is only 1DOF per node) that are 'free' and 'supported', respectively. As an example, the vectors for the example in part (b) are shown below. But you will have to write an algorithm that does this automatically for any inputs.

id_s = [1;2;3;4;6;7;9;10;11;12]
id_f = [5;8]

Tip: This naming convention is based on mechanics problems. For Prandtl stress problems, the perimeter nodes are 'supported', hence they have unknown 'forces' $\{P_s\}$, and known 'displacements' $\{d_s\} = \{0\}$). This is because the $\{d\}$ vector represents the nodal values of ϕ, which is zero on the boundary. The nodes in the interior are 'free', hence they have known 'forces' $\{P_F\} = \{0\}$, and unknown displacements $\{d_F\}$. Note that the $\{P\}$ vector is globally a boundary integral, and hence is zero on all interior nodes. We will also add the $\{P_{FEF}\}$ later, which also has S and F parts, but will not be zero anywhere. Note that you could actually solve the $\{P_S\}$ 'reaction forces', but we won't need them.

Discussion 1: For heat conduction, fluid flow, and 2D/3D elasticity problems, the boundary conditions can be a bit trickier, as they can be either displacement/temp/pressure type (which is the $\{d\}$ vector) or they can be stress/flux type (which is the $\{P\}$ vector, note these all have to do with derivatives of displacement). Commercial software allows you to select various surfaces (or nodes directly) and define different kinds of boundary conditions, and a lot of code behind the scenes to take that graphical information and construct the id_f and id_s vectors (though they may use other names).

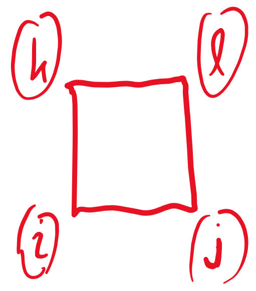
Discussion 2: You might be thinking that is not a lot of free nodes for this example. That is correct, this is way too coarse to get a good result.

Discussion 3: Generally, BCs are a user input option. However, for Prandtl stress, the BCs are always 0 on the perimeter so it can be 'hard-coded' without giving the user a chance to change it (because they should not change it).

(d) ***Assemble Stiffness Equations:*** This part of the code will construct the global $[\boldsymbol{K}]$ and $\{\boldsymbol{P_{FEF}}\}$. This part is the core of any FEM code. It is also a few different parts: (i) calculating $[K^e]$ and $\{P^e_{FEF}\}$ for each element, and (ii) assembling the global versions. This is all done with a loop. The steps for $[K]$ is shown in pseudo-code below, but $\{P_{FEF}\}$ is similar, but will need θ as an input as that is the 'body force' term for Prandtl stress.

```
K = zeros(n_dof, n_dof);      % initialize Global [K]
for e = 1 : n_ele
    collect node numbers for current element
    i_node = e_info(e,1);
    j_node = e_info(e,2);
    k_node = e_info(e,3);
    l_node = e_info(e,4);
    x_i = n_info(i_node,1)
    y_i = n_info(i_node,2)
    x_j = ...
    :
    [Be] = function of stuff you know
    [ke] = see eq. sheet = A   4x4
    k(i_node,i_node) = k(i_node,i_node) + ke(1,1)
    k(i_node,j_node) = k(i_node,j_node) + ke(1,2)
        :  All 16 terms, there are better ways
end
```

Appeding to make sure no overwriting



(e) ***Create the* {*P*} *vector:*** This part is trivial for Prandtl Stress problems (but not necessarily for other problems), just make $\{P\}$ a vector of zeros.

Discussion 1: For heat conduction or flow problems, if there are any point sources/sinks within the domain of the problem, those would need to be put into the $\{P_F\}$ part of the $\{P\}$ vector (not the $\{P_S\}$ part though, that is something that will be solved). This is equivalent to point loads for structures problems. For Prandtl Stress this is just a vector of zeros (though I hope you think about why). In fact, you can make the entire $\{P\}$ vector all zeros, because as we will see later, we won't use the $\{P_S\}$ part anyway.

Discussion 2: Note the fundamental difference between: (i) $[K]$ and $\{P_{FEF}\}$, and (ii) $\{P\}$. The former quantities are integrals over the element domain, while the $\{P\}$ vector comes from integrals around the perimeter, or it may also be literal point sources/sinks at the nodes.

(f) ***Partition Equations:*** I give you the answer below. This is nothing more than stacking the free and supported rows and columns of the stiffness equations as follows. See part (g) to see why we need these parts. Matlab syntax is quite efficient at selecting rows and columns of a stiffness matrix.

All zeros

```
Pf = P(idf);
Pfeff = Pfef(idf);
Kff = K(idf : idf);
```

All we need

More Generally, would also need:

$\{d_s\}$ = some defined/known disp.

However, for Prandtl, $\{d_s\} = \{0\}$ = ϕ on BDRY

so don't need it

(g) ***Solve:*** I also give the code for this part.

$$d_F = \text{inv}(k_{ff}) \times P_{feFe} \quad \Big\} \text{ all you need}$$

More generally, we would solve the full system below

$$\begin{Bmatrix} \{P_{FEF} \; F\} \\ \{P_{FEF} \; S\} \end{Bmatrix} + \begin{Bmatrix} \overset{\text{0 for Prandtl}}{\cancel{\{P_F\}}} \\ \underset{\text{secondary unknowns}}{\{P_S\}} \end{Bmatrix} = \begin{bmatrix} [k_{FF}] & [k_{FS}] \\ [k_{SF}] & [k_{SS}] \end{bmatrix} \begin{Bmatrix} \overset{\text{primary unknowns}}{\{d_F\}} \\ \underset{\text{0 for Prandtl } \phi \text{ on BDRY}}{\cancel{\{d_S\}}} \end{Bmatrix}$$

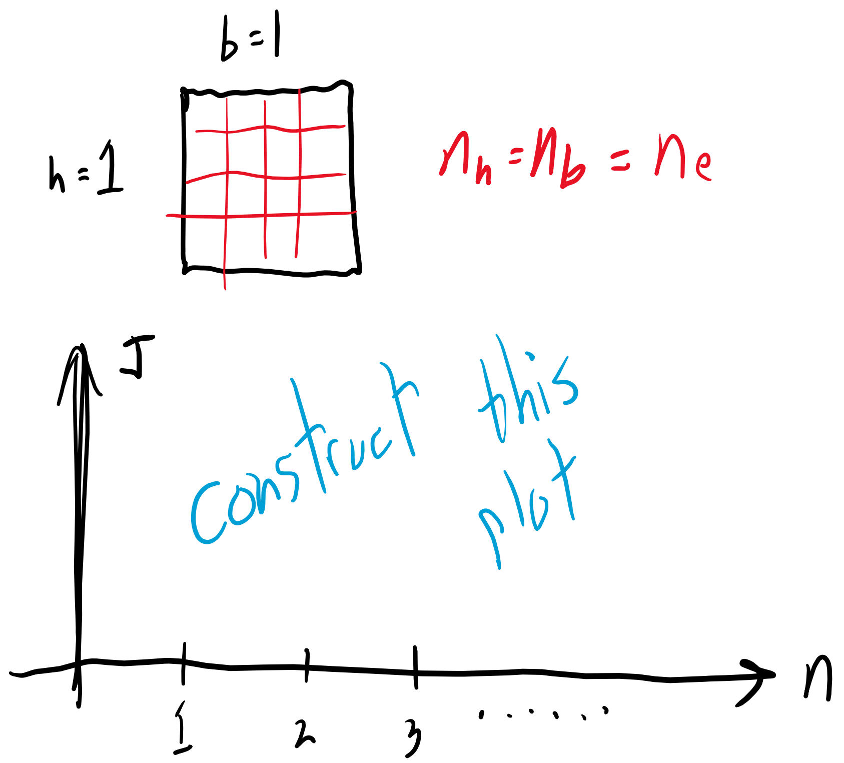
(h) ***Reconstruct $\{d\}$:*** It is often useful to reconstruct the whole $\{d\}$ vector, and it will be useful in part (g). This is often useful for post-processing/plotting. Please put the whole vector back together. This is about re-inserting the 0s (or $\{d_s\}$ more generally) in their proper place from the boundary nodes. There is many ways to do this, I recommend using the idf and ids vectors to do this in two lines of code, but you could manually loop through all components instead if you prefer.

(i) ***Post-Processing to get J:*** Create an algorithm that calculates the torsion constant using the equation below. The J value should be the main output of your code. The only challenge here is finding the volume under the ϕ curve. This will use the $\{d\}$ vector. I will leave it up to you how you do this. You could use basic tributary areas (i.e., multiply d_i at a node by the tributary areas of the nodes), or you could formally integrate the area under ϕ on each element (you have the shape functions to do this). Both would give similar results if the mesh is fine enough. These kinds of decisions on how to integrate come up a lot in FEM (including the integrals needed to get the stiffness equations). The first type of integration is "intuitive" and is one kind of "lumped" approach. The latter is called a "consistent" approach as it is fully consistent with the governing equations.

$$J = \frac{2\int_A \phi \, dA}{G\theta}$$

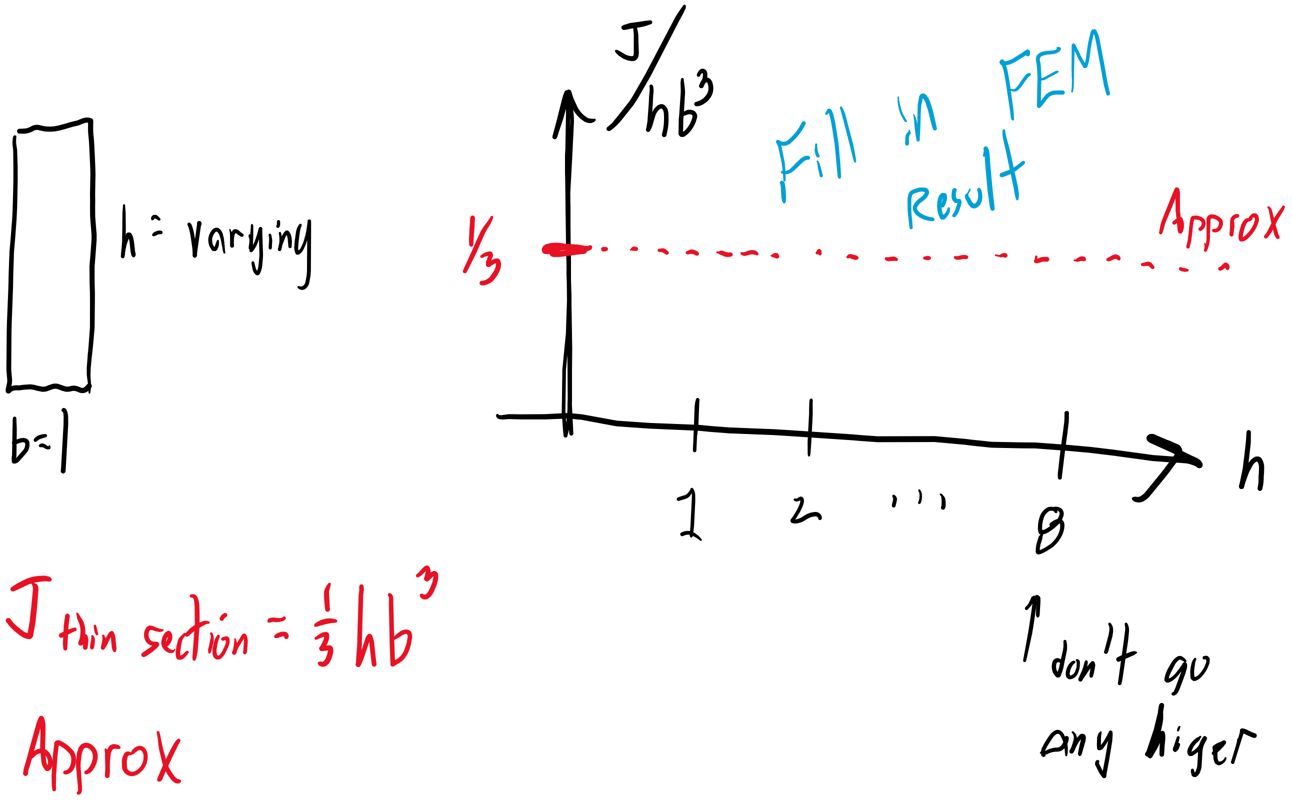
3. ***Exploring Torsion:*** Use your code to do the following:

(a) ***Convergence Study:*** Plot J vs. mesh density. A convergence study is usually one of the first things you need to do, you are trying to get a handle of the mesh density needed to get converged results for the problem of interest. Let's simplify this to the case for a square cross section as shown below, where you simultaneously increase the mesh density horizontally and vertically. In general, "Mesh Density" is not really unique, as we could have different numbers of elements in different directions, or different element types, or varying density across the domain, and/or strange domain shapes. Thus, convergence studies usually involve some measure of result (for us this is J) vs. some measure of density.

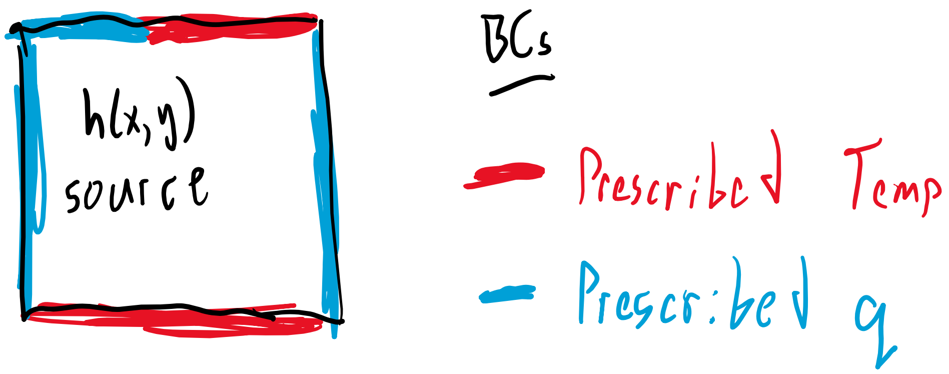


select any G & θ, you will see it does not affect J

(b) ***(NOT REQUIRED, Please explore if you have time)*** Exact vs. Approximate Approaches: Generate the plot below for cross-sections of changing aspect ratio. For each analysis you do, you will have to do a convergence study of your own (do not attach it) to determine whether your mesh is sufficient. You will also have to make sure that your elements are reasonably close to squares, so your nb and nh may not be the same value.



4. ***2D Diffusion Problems:*** We will not code up a solver for diffusion problems. However, in this question, we walk through all of the parts of question 2, and briefly describe how your Prandtl Stress code would need to be modified to instead solve the heat transfer problem below. I am showing what really detailed answers would be for parts (a) and (b), please fill in the rest (I am not expecting as much detail, just discuss what you think it would be needed).



(a) The material property would now be the conduction coefficient k, instead of G. Also, the body force term would now have h(x,y) details instead of just θ. If the source was a function of position instead of a constant, this would be a lot more involved than what we did for torsion. Also, the boundary conditions cannot really be hard-coded into the latter steps any more. It looks like the user should be able to define where the red and blue parts of the boundary are, and what those prescribed quantities are. The later parts of the code should take that information to build idf and ids, and also to fill in $\{d_S\}$ and $\{P_F\}$.

(b) Fortunately, the mesher is unchanged. It should work in the same way. A caveat here could be that sometimes various things can be stored in n_info or e_info (though that is not the only way to do it). For example, since the boundary now has different types of BCs (temp type and flux type), it might be good to add another column to n_info that flags which type each node is. Another example is that h(x,y) could be defined as piecewise constant on elements (so able to vary across the domain, but constant within each element), and thus the h within an element could be added as another column of e_info. Lastly, if material properties vary with position, that is often put into e_info as well.

(c) …

(d) … (Note: Please discuss common form of governing equations for this & Prandtl)

(e) …

(f) …

(g) …

(h) …

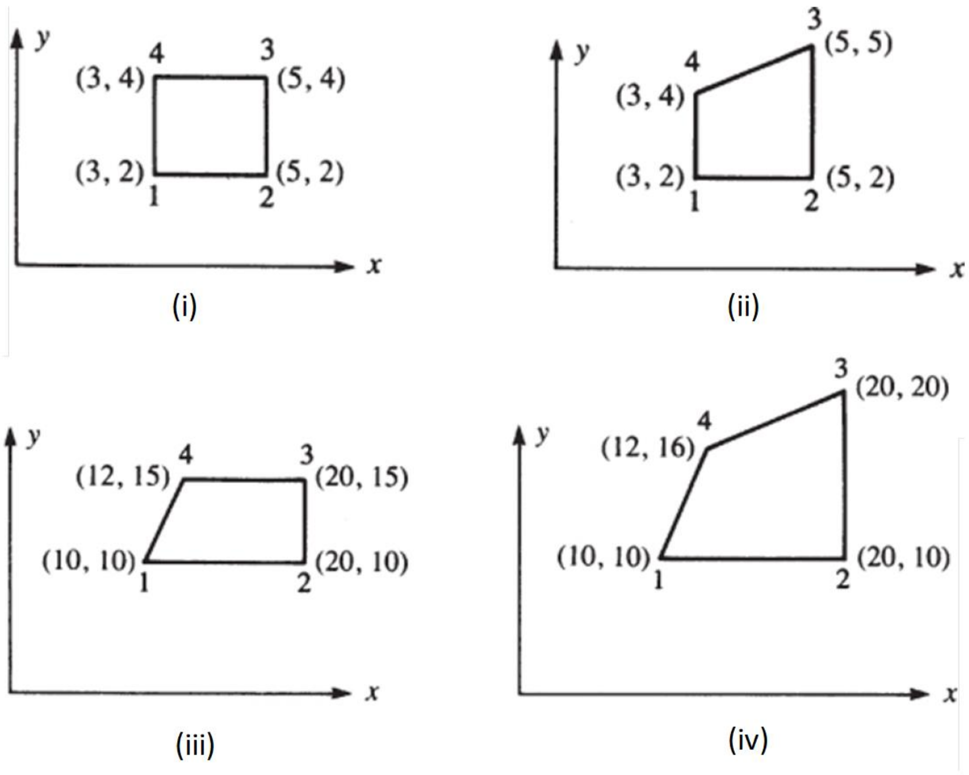
(i) …

5. Briefly describe why the meshing becomes a bit more involved for 2D and 3D elasticity than for the above problems. ***Hint:*** Think of the difference between nodes & DOFs (the difference is hard to see when you have scaler fields, e.g., T or P.

You are not required to turn in the two problems below this (the solution for these is also already posted), but you are responsible for this material on the midterm.

6. ***Isoparametric Elements (Do Not Turn In, Solution Available):*** The main difference in implementing isoparametric elements is that the $[K^e]$ and $\{P^e_{FEF}\}$ components are evaluated in the η, ξ space instead of the physical space. Once you have the components, they can be assembled as with any other elements.

   (a) Sketch vertical & horizontal differential strips on the 4 different isoparametric elements. Use this to show why integration in the original space would be a pain, especially for horizontal strips on (ii), vertical strips on (iii), and either way on (iv).
   (b) Determine the Jacobian matrix and its determinant for the four different elements.



Discussion 1: If you swapped the rectangular elements in your Prandtl Stress code for isoparametric elements, you could handle irregular element shapes. However, that would also require some more general meshing. These elements are much better for representing irregular (i.e., not rectangular) cross-sections. Commercial FEM software automatically decides which elements to use, and will also mix and match within a problem. That is a big algorithm too.

Discussion 2: Just a cool note. You can map from circles to straight lines. This allows for things like having a curved surface on the real part, but exactly capturing it in the FEM with straight vertex formulations. This is a better way than making a curvy shape with a bunch of straight lines.

7. ***Discrete Integration (Do Not Turn In, Solution Available)***

In class, we noted that Gaussian quadrature can represent exactly the integral of a polynomial of order $2n$-1, and we ~~found the~~ solved for the appropriate locations, ξ_k, and weights, W_k, for two-point quadrature. Show that, for three-point quadrature, the appropriate locations are $\xi = \left\{-\sqrt{3/5}, 0, \sqrt{3/5}\right\}$ and the corresponding weight functions are $W = \{5/9, 8/9, 5/9\}$ by solving the integral:

$$I = \int_{-1}^{1} f(\xi)\, d\xi = \sum_{k=1}^{3} W_k f(\xi_k) = W_1 f(\xi_1) + W_2 f(\xi_2) + W_3 f(\xi_3)$$

$$f(\xi) = a_0 + a_1\xi + a_2\xi^2 + a_3\xi^3 + a_4\xi^4 + a_5\xi^5$$

Use two and three point quadrature to solve the following integrals and comment on your results.

$$\int_{-1}^{1} \cos\frac{s}{2} ds \qquad \int_{-1}^{1} s^2 ds \qquad \int_{-1}^{1} s^4 ds$$

Discussion 1: Given that one of the biggest steps in any FEM solver is integration over elements (which have been 2D in this assignment, but will also be 3D later). These integrals can be rather tedious (even when using isoparametric coordinates), especially if the shape functions are complex. Sometimes, we can do these integrals generically, and yield a general equation. For example, in 1D elasticity, AE/L always appears. Another example is that for constant strain triangles, the integrand is constant, and we were also able to get a general equation for $[K^e]$, though $\{P_{FEF}\}$ is more involved since $[N]$ is not constant. However, as you are starting to see, sometimes we don't derive a general 'stencil', and instead have to evaluate the integrals for each element during assembly. The tedium, and computational expense of this operation has led to things like Guassian quadrature.

Discussion 2: Interestingly, discrete integration methods have resulted in some perhaps unexpected benefits in accuracy (but also costs). We will talk about this later when we discussed reduced integration.