UC Berkeley Math 228B, Spring 2019: Problem Set 3

Due March 7

1. Write a Julia function with the syntax

```
p, t, e = pmesh(pv, hmax, nref)
```

which generates an unstructured triangular mesh of the polygon with vertices `pv`, with edge lengths approximately equal to $h_{\max}/2^{n_{\text{ref}}}$, using a simplified Delaunay refinement algorithm. The outputs are the node points `p` (N-by-2), the triangle indices `t` (T-by-3), and the indices of the boundary points `e`.

(a) The 2-column matrix `pv` contains the vertices x_i, y_i of the original polygon, with the last point equal to the first (a closed polygon).

(b) First, create node points along each polygon segment, such that all new segments have lengths $\leq h_{\max}$ (but as close to $h_{\max}$ as possible). Make sure not to duplicate any nodes.

(c) Triangulate the domain using the `delaunay` function in the mesh utilities.

(d) Remove the triangles outside the domain (see the `inpolygon` command in the mesh utilities).

(e) Find the triangle with largest area A. If $A > h_{\max}^2/2$, add the circumcenter of the triangle to the list of node points.

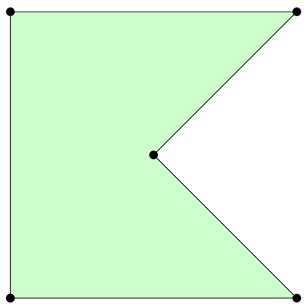
(f) Retriangulate and remove outside triangles (steps (c)-(d)).

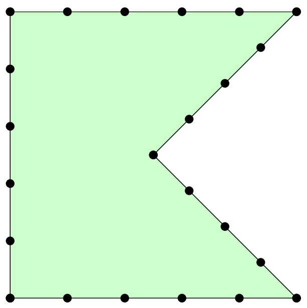
(g) Repeat steps (e)-(f) until no triangle area $A > h_{\max}^2/2$.

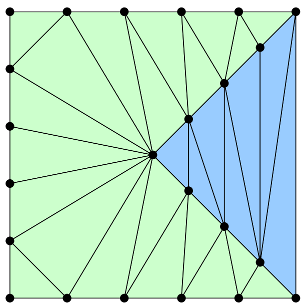
(h) Refine the mesh uniformly n_{ref} times. In each refinement, add the center of each mesh edge to the list of node points, and retriangulate. Again, make sure not to duplicate any nodes, using e.g. the command `unique(p, dims=1)`.

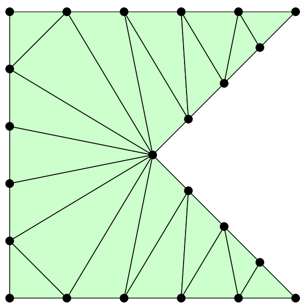
Finally, find the nodes `e` on the boundary using the `boundary_nodes` function. The following commands create the example in the figures. Also make sure that the function works with other inputs, that is, other polygons, $h_{\max}$, and n_{ref}.

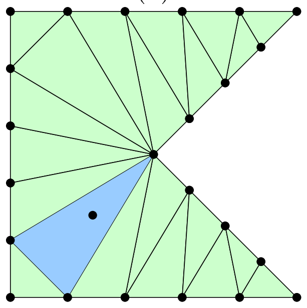
```
pv = [0 0; 1 0; .5 .5; 1 1; 0 1; 0 0]
p, t, e = pmesh(pv, 0.2, 1)
tplot(p, t)
```

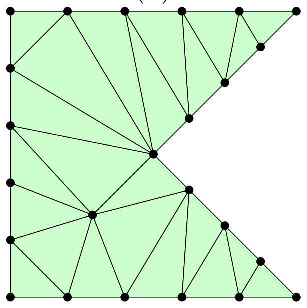

(a)

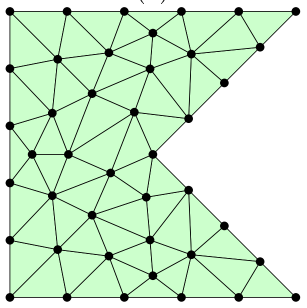

(b)

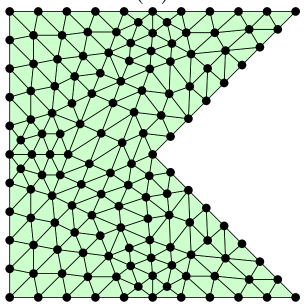

(c)


(d)


(e)


(f)


(g)


(h)

2. Consider the Eikonal equation for first arrivals/optimal path planning problems. The equation can be written

$$F(x,y)|\nabla\phi(x,y)| = 1,$$

where $F(x,y)$ is the speed function. We will use the solution $\phi(x,y)$ to determine the optimal path from the departure point (x_D,y_D) to the arrival point (x_A,y_A), where $\phi(x_A,y_A) = t$. The level set with value t of the function $\phi(x,y)$ gives the maximum distance away from our departure point that can be traveled in a time t. In addition, the optimal path between the departure point and the arrival point is determined by traveling in the normal direction of the level sets.

We will solve the Eikonal equation using a level set method and a time-stepping approach. The equation

$$\phi_t + F|\nabla\phi| = 1, \qquad \phi(x_D,y_D) = 0$$

is integrated in time until a steady-state is reached. This is not an efficient method for solving the Eikonal equation, but it will be sufficient for this problem set (and it illustrates how to solve more general time-dependent problems). If you are interested in more sophisticated solvers, feel free to implement the more efficient fast marching method instead.

(a) Write a computer code to solve the Eikonal equation on the unit square $x,y \in [0,1]$. Use the first-order upwinded scheme in space, and appropriate treatment of the boundaries.

(b) Write a computer code to find the optimal path between the departure/arrival point, by solving the ODE $d\boldsymbol{r}/dt = \boldsymbol{n}$, where $\boldsymbol{r}$ is the current position on the path and $\boldsymbol{n}$ is the normal vector.

(c) Run your codes with grid spacing $h = 1/100$ for the following cases and plot both the solutions (e.g. as contour curves of $\phi(x,y)$) and the optimal paths:

Case 1: $(x_D,y_D) = (0.2,0.2)$, $(x_A,y_A) = (0.8,0.8)$, $F(x,y) = 1$ (for testing).

Case 2: $(x_D,y_D) = (0.2,0.2)$, $(x_A,y_A) = (0.8,0.45)$, and

$$F(x,y) = \begin{cases} 1.0 & \text{if } y \geq 0.5, \\ 0.5 & \text{if } y < 0.5. \end{cases}$$

Case 3: $(x_D,y_D) = (0.2,0.2)$, $(x_A,y_A) = (0.8,0.8)$, and

$$F(x,y) = 1 - 0.9 \cdot \cos(4\pi x) \cdot e^{-10\,((x-.5)^2+(y-0.5)^2)}$$

Case 4: Make up your own speed function F, only returning the values 0.01 and 1 but with a non-trivial optimal path.

Code Submission: Submit a zip-file on bCourses which contains a Julia file named `pmesh.jl` where all requested functions are defined (that is, `pmesh`), and any other supporting functions. Alternatively, submit a Jupyter notebook which will define this function when executed.