Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2024

Project 3 - Triangular mesh generator

Due Friday, March 22

First we include some libraries and define utility functions from the lecture notes:

In [1]:
```
using PyPlot, LinearAlgebra, Delaunator

function tplot(p, t)
    # Plot triangular mesh with nodes `p` and triangles `t`
    tris = convert(Array{Int64}, hcat(t...)')
    tripcolor(first.(p), last.(p), tris .- 1, 0*tris[:,1],
              cmap="Set3", edgecolors="k", linewidth=1)
    axis("equal")
    return
end

# Delaunay triangulation `t` of array of nodes `p`
delaunay(p) = collect.(triangulate(p).triangles)
```

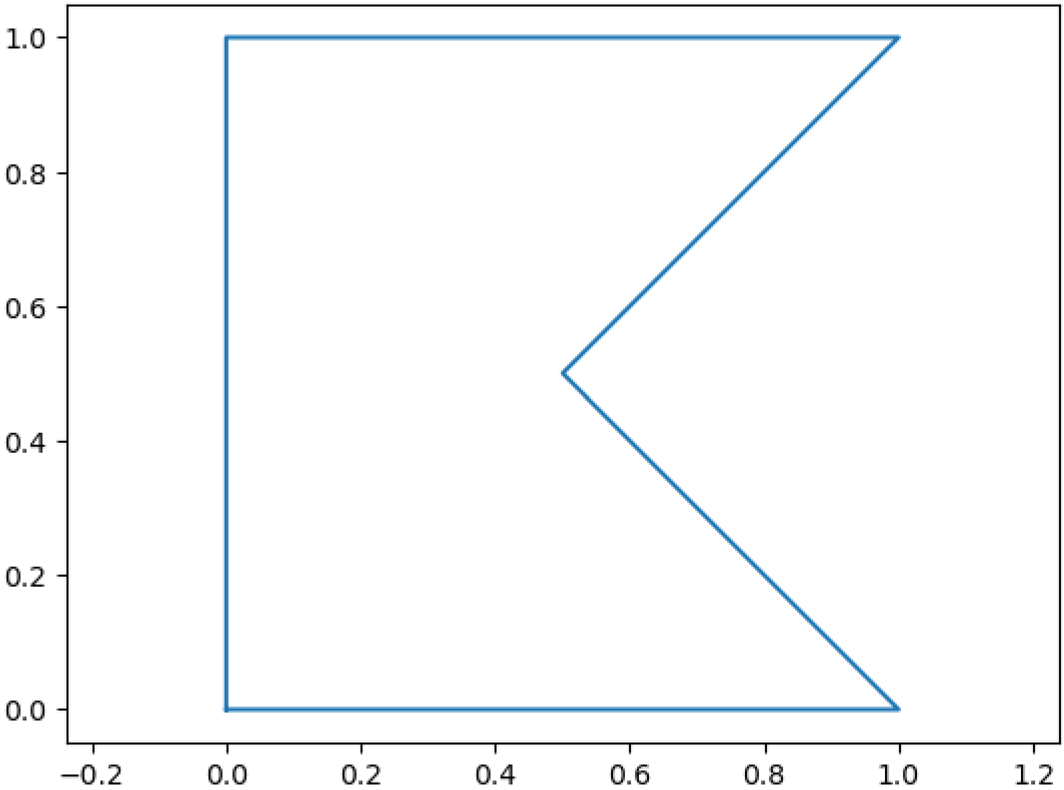
```
[ Info: Precompiling Delaunator [466f8f70-d5e3-4806-ac0b-a54b75a91218]
```

Out[1]:
```
delaunay (generic function with 1 method)
```

Description

In this project you will write an unstructured triangular mesh generator based on the Delaunay refinement algorithm. The steps will be described in detail, and for testing we will use the following simple polygon:

In [2]:
```
pv = [[0,0], [1,0], [0.5,.5], [1,1], [0,1], [0,0]]
plot(first.(pv), last.(pv))
axis("equal");
```



Problem 1 - Point in polygon

Write a function `inpolygon(p, pv)` which determines if a point `p` is inside the closed polygon `pv`. For example, in the test polygon above, the point $(0.6, 0.3)$ is inside but $(0.8, 0.3)$ is outside. For the algorithm, use the "Crossing number method" as described here: https://observablehq.com/@tmcw/understanding-point-in-polygon.

In [3]:

```
function inpolygon(p, pv)
    x, y = p

    inside = false
    for i in 1:length(pv)
        xi, yi = pv[i]
        j = i == 1 ? length(pv) : i - 1
        xj, yj = pv[j]

        intersect = (yi > y) ⊻ (yj > y) && (x < (xj - xi) * (y - yi) / (y
        if intersect
            inside = !inside
        end
    end

    return inside
end
```

Out[3]: `inpolygon (generic function with 1 method)`

Problem 2 - Triangle properties

Next we need functions for computing some basic quantities from triangles. Here, a triangle `tri` is represented as an array of 3 points, e.g.

In [6]:
```
tri = [[1,0.5], [2,1], [0,3]]
```

Out[6]:
```
3-element Vector{Vector{Float64}}:
 [1.0, 0.5]
 [2.0, 1.0]
 [0.0, 3.0]
```

Problem 2(a) - Triangle area

Write a function `tri_area(tri)` which returns the area of `tri`.

In [4]:
```
function tri_area(tri)
    return 0.5*abs(tri[1][1]*(tri[2][2] - tri[3][2]) + tri[2][1]*(tri[3][
end
```

Out[4]:
```
tri_area (generic function with 1 method)
```

Problem 2(b) - Triangle centroid

Write a function `tri_centroid(tri)` which returns the centroid of `tri` (https://en.wikipedia.org/wiki/Centroid#Of_a_triangle).

In [8]:
```
function tri_centroid(tri)
    return sum(tri)/3
end
```

Out[8]:
```
tri_centroid (generic function with 1 method)
```

Problem 2(c) - Triangle circumcenter

Write a function `tri_circumcenter(tri)` which returns the circumcenter of `tri` (https://en.wikipedia.org/wiki/Circumcircle#Cartesian_coordinates_2).

In [14]:
```
function tri_circumcenter(tri)
    Ax, Ay = tri[1]
    Bx, By = tri[2]
    Cx, Cy = tri[3]
    D = 2*(Ax*(By - Cy) + Bx*(Cy - Ay) + Cx*(Ay - By))
    return [((Ax^2 + Ay^2)*(By - Cy) + (Bx^2 + By^2)*(Cy - Ay) + (Cx^2 +
            ((Ax^2 + Ay^2)*(Cx - Bx) + (Bx^2 + By^2)*(Ax - Cx) + (Cx^2 +
end
```

Out[14]:
```
tri_circumcenter (generic function with 1 method)
```

Problem 3 - Mesh generator

Write a function with the syntax `p,t = pmesh(pv, hmax)` which generates a mesh `p,t` of the polygon `pv`, with triangle side lengths approximately `hmax`. Follow the algorithm as described below.

(a) The input `pv` is an array of points which defines the polygon. Note that the last point is equal to the first (a closed polygon).

(b) First, create node points `p` along each polygon segment, separated by a distance approximately equal to `hmax`. Make sure not to duplicate any nodes.

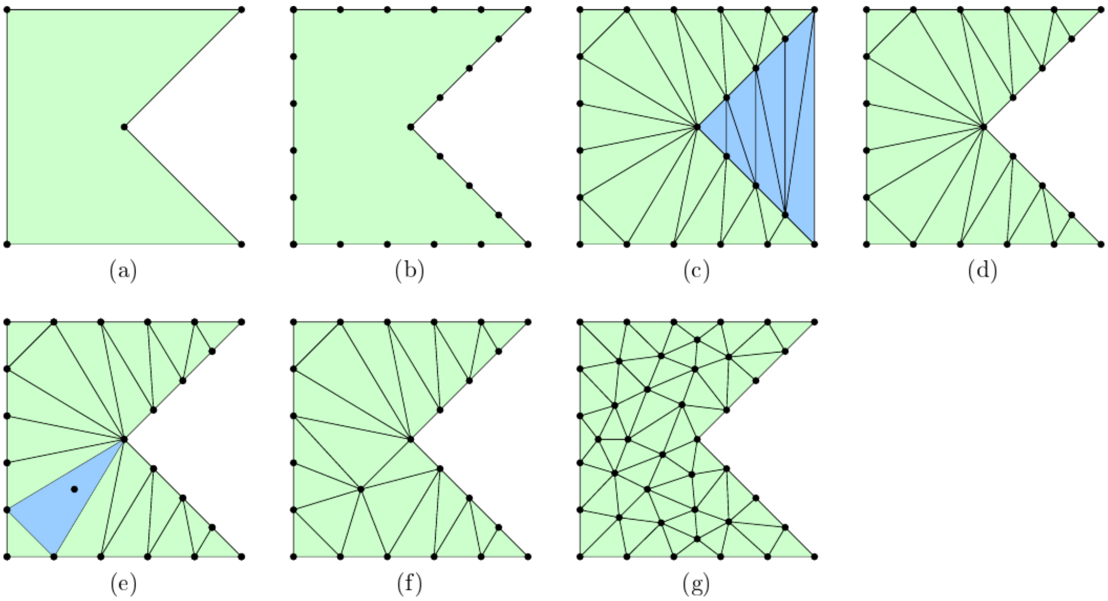
(c) Triangulate the domain using the `delaunay` function.

(d) Remove the triangles outside the polygon, by computing all the triangle centroids (using `tri_centroid`) and determining if they are inside (using `inpolygon`).

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

(f) Repeat steps (c)-(d), that is, re-triangulate and remove outside triangles.

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



In [120…

```
function pmesh(pv, hmax)
    # a
    n = length(pv)

    # b
    nodes = []
    for i in 1:n-1
        x1, y1 = pv[i]
        x2, y2 = pv[i + 1]

        segment_length = sqrt((x2 - x1)^2 + (y2 - y1)^2)
        num_nodes = Int(round(segment_length / hmax))
        x_diff = (x2 - x1) / num_nodes
        y_diff = (y2 - y1) / num_nodes

        x, y = x1, y1
        for k in 1:num_nodes
            push!(nodes, [x, y])
            x += x_diff
            y += y_diff
        end
```

```
        push!(nodes, [x2, y2])
    end
    p = collect(nodes)
    t = []

    while true
        # c
        t = delaunay(p)
        # d
        t = [i for i in t if inpolygon(tri_centroid(p[i]), pv)]
        # e
        areas = [tri_area(p[i]) for i in t]
        if maximum(areas) > hmax^2/2
            push!(p, tri_circumcenter(p[t[argmax(areas)]]))
        else
            break
        end
    end

    return p, t
end
```

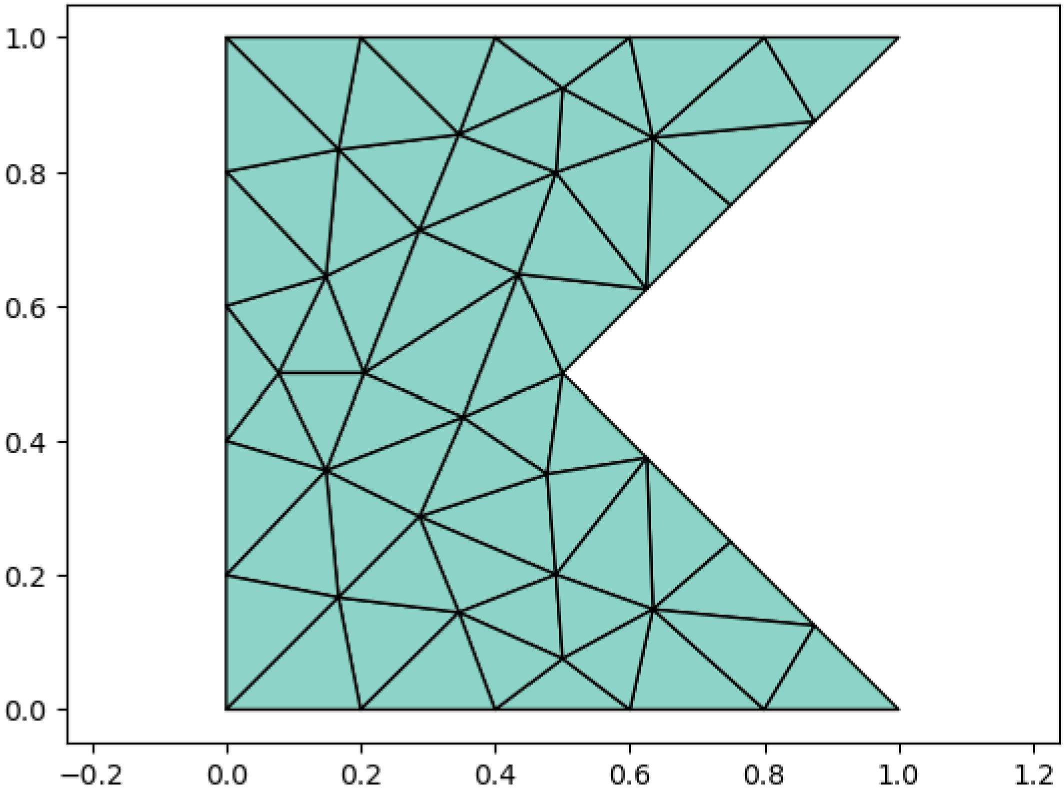
Out[120]: pmesh (generic function with 1 method)

Test cases

Run the cases below to test your mesh generator.

In [121…

```
# The polygon in the examples
pv = [[0,0], [1,0], [0.5,.5], [1,1], [0,1], [0,0]]
p,t = pmesh(pv, 0.2)
tplot(p,t)
```



In [122…

```
# A more complex shape
pv = [[i/10,0.1*(-1)^i] for i = 0:10]
append!(pv, [[.5,.6], [0,.1]])
p,t = pmesh(pv, 0.04)
tplot(p,t)
```