Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2023

Project 3 - Triangular mesh generator

Due Friday, March 24

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

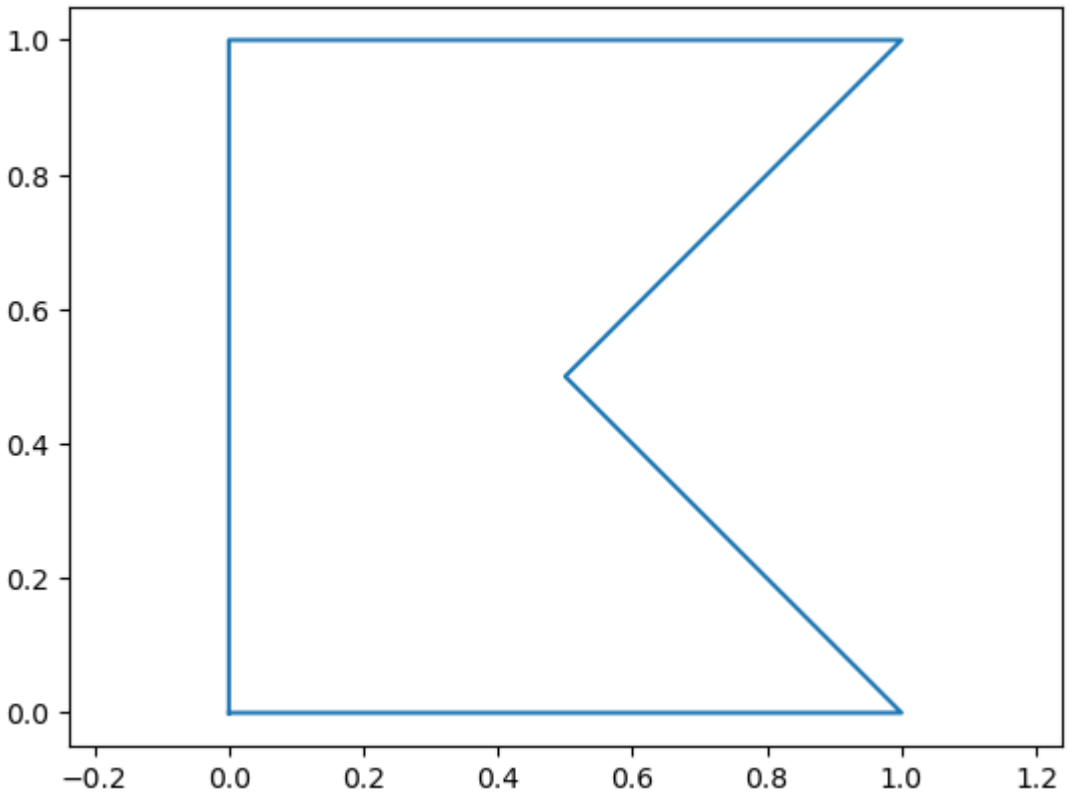
In [1]:
```
using PyPlot, LinearAlgebra, PyCall

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

function delaunay(p)
    # Delaunay triangulation `t` of array of nodes `p`
    tri = pyimport("matplotlib.tri")
    t = tri[:Triangulation](first.(p), last.(p))
    t = Int64.(t[:triangles] .+ 1)
    t = [ t[i,:] for i = 1:size(t,1) ]
end
```

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 (https://observablehq.com/@tmcw/understanding-point-in-polygon).

In [3]:
```
function inpolygon(p,pv)
    x=p[1]
    y=p[2]
    inside=false;
    for i=1:length(pv)
        if i==length(pv)
            j=1
        else
            j=i+1
        end
        xi=pv[i][1]
        yi=pv[i][2]
        xj=pv[j][1]
        yj=pv[j][1]
        intersect=((yi>y)!=(yj>y)) && (x<(xj - xi) * (y - yi) / (yj -
        if (intersect)
            inside = !inside;
        end
    end
    return inside;
end

```

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

In [4]:
```
inpolygon([.6,.5],pv)
```

Out[4]:
```
false
```

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 [5]:
```
tri = [[1,0.5], [2,1], [0,3]]
```

Out[5]:
```
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 [6]:
```
function tri_area(tri)
    x1,y1=tri[1][1],tri[1][2]
    x2,y2=tri[2][1],tri[2][2]
    x3,y3=tri[3][1],tri[3][2]
    area=abs(0.5*((x1*(y2-y3))+(x2*(y3-y1))+(x3*(y1-y2))))
    return area
end


```

Out[6]:
```
tri_area (generic function with 1 method)
```

In [7]:
```
tri_area(tri)
```

Out[7]:
```
1.5
```

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 (https://en.wikipedia.org/wiki/Centroid#Of_a_triangle)).

In [8]:
```
function tri_centroid(tri)
    x1,y1=tri[1][1],tri[1][2]
    x2,y2=tri[2][1],tri[2][2]
    x3,y3=tri[3][1],tri[3][2]
    x=(x1+x2 +x3)/3
    y=(y1+y2+y3)/3
    centroid=[x,y]
end
tri_centroid(tri)
```

Out[8]:
```
2-element Vector{Float64}:
 1.0
 1.5
```

Problem 2(c) - Triangle circumcenter

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

In [9]:

```
function tri_circumcenter(tri)
    x1,y1=tri[1][1],tri[1][2]
    x2,y2=tri[2][1],tri[2][2]
    x3,y3=tri[3][1],tri[3][2]
    d = 2 * (x1 * (y2 - y3) + x2* (y3 - y1) + x3 * (y1 - y2))
    ux = ((x1^2 + y1^2) * (y2 - y3) + (x2^2 + y2^2) * (y3 - y1) + (x3^
    uy = ((x1^2 + y1^2) * (x3 - x2) + (x2^2 + y2^2) * (x1 - x3) + (x3^
    return ux,uy
end
tri_circumcenter(tri)


```

Out[9]: (0.9166666666666666, 1.9166666666666667)

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.

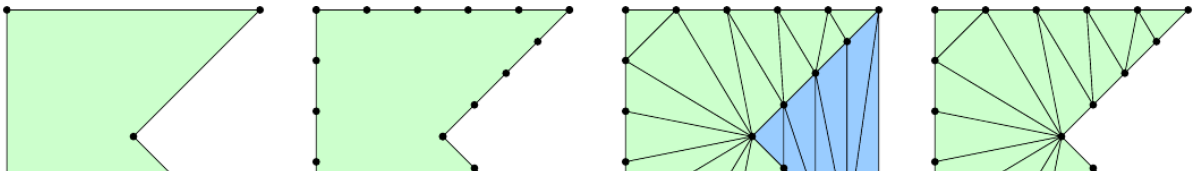
(c) Triangulate the domain using the `delaunay` function.

(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 [10]:
```
function dist(x1,y1,x2,y2)
    sqrt((x2-x1)^2+(y2-y1)^2)
end
```

Out[10]:
```
dist (generic function with 1 method)
```

In [11]:
```
pv = [[0,0], [1,0], [0.5,.5], [1,1], [0,1], [0,0]]
```

Out[11]:
```
6-element Vector{Vector{Float64}}:
 [0.0, 0.0]
 [1.0, 0.0]
 [0.5, 0.5]
 [1.0, 1.0]
 [0.0, 1.0]
 [0.0, 0.0]
```

In [ ]:

```
function (pv,hmax)
    X1,Y1= pv[1][1], pv[1][2]
    X2,Y2= pv[2][1], pv[2][2]
    X3,Y3= pv[3][1], pv[3][2]
    X4,Y4= pv[4][1], pv[4][2]
    X5,Y5= pv[5][1], pv[5][2]
    l1=dist(X1,Y1,X5,Y5)
    l2=dist(X5,Y5,X4,Y4)
    l3=dist(X4,Y4,X3,Y3)
    l4=dist(X3,Y3,X2,Y2)
    l5=dist(X2,Y2,X1,Y1)
    N1=1+(l1/hmax)
    N2=1+(l2/hmax)
    N3=1+(l3/hmax)
    N4=1+(l4/hmax)
    N5=1+(l5/hmax)
    for i=1:n[X1]
        p=[N1,N2,N3,N4,N5]
        delauney(p)
         for tri in t
            tcent=tri_centroid(t)
            if inpolygon(tcent)=False
                remove!(t, false)
                return t
                for tri in t
                    A=max(tri_area(t))
                    if A > (hmax^2)/2
                        mc=tri_circumcenter(maxar)
                        return push!(p,mc)
```

In [9]:

```
s1= pv[1:2]
s2=pv[2,4]
s3=pv[4,3]
s4=pv[4:5]
s5=pv[5:6]
```

Out[9]:

```
2-element Vector{Vector{Float64}}:
 [0.0, 1.0]
 [0.0, 0.0]
```

Test cases

Run the cases below to test your mesh generator.

In [ ]:

```
# 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 [ ]:

```
# 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)
```