Random constructions in the plane
Imperial College summer research supervised by Dr. Davoud Cheraghi

Pantelis Tassopoulos

Summer 2022

Contents

1 Overview

More specifically, I studied the notion of harmonic measure in the plane, its various formulations involving conformal maps and Brownian motion, culminating in the study of the so-called conformally balanced trees following work from Professor of mathematics at Stony Brook Christopher Bishop. This was a very profitable experience as it helped me further refine my analytical problem-solving and decomposition skills due to the nature of the work in the project. In conjunction with the above, my my communication and organisational skills were invariably improved as I engaged in weekly meetings with my supervisor Dr. Cheraghi, wherein I discussed the progress of the project and received feedback on approaches to obstacles, incorporating said suggestions into the project. I obtained a lot of insight into the world of academia and the way research is conducted.

2 Conformally Balanced Trees

- mention paper of Chris Bishop
- mention reading books, lecture notes, geometric function theory etc.
- Conjecture regarding Hausdorff distance and tree like structure (Conformally balanced trees have a unique embedding into the plane after undergoing hydrodynamic stabilisation –¿ explain maybe)
- mention zipper algorithm and what it does and how it was used to test the above hypothesis in a simple fashion

Degree three vertex Tree generation algorithm

```
import numpy as np
#Nesterov Potential
n = 256
x = np.linspace(-1.5, 1.2, n)
y = np.linspace(-0.2, 2, n)
X, Y = np.meshgrid(x, y)

intervals = np.arange(1, 1e5, 20)

ntraj = 1000
# Initialize holder for trajectories
colors = plt.cm.jet(np.linspace(0,1,np.minimum(ntraj, 7)))
minima_nesterov = []
for j in tqdm(range(ntraj)):
    points_x, points_y = train_nesterov(intervals,
    learning_rate = 1e-4, a = 1, tolerance = 1e-5)
    minima_nesterov.append(MB_potential(points_x[-1],points_y[-1]))
    if j <=6:
        plt.scatter(points_x, points_y, color = colors[j], s = 0.1)
        for i in range(len(points_x)-1):
            plt.annotate('', xy =[points_x[i+1],
            points_y[i+1]], xytext= [points_x[i], points_y[i]],
                        arrowprops={'arrowstyle': '->', 'color': colors[j],
'lw': 1},
                        va='center', ha='center')

plt.contour(X, Y, vMB_potential(X, Y).clip(max=200), 8,
alpha=.75, cmap='viridis')
C = plt.contour(X, Y, vMB_potential(X,Y).clip(max=200), 8)
plt.title('Muller-Brown-potential-with-Nesterov-accelerated-GD')
plt.clabel(C, inline=1, fontsize=10)
plt.xticks([])
plt.yticks([])
#plt.legend()
```

Line Perturbation algorithm

```
import numpy as np
#Line perturbation (10 vertices)

import random
import numpy as np
from scipy.spatial import ConvexHull
from scipy.spatial.distance import directed_hausdorff
```

```
import matplotlib.pyplot as plt


#to contain file names
#files = ['_base', '_line1', '_line2', '_line3', '_line4', '_line5', '_line6', '_lin
Hausdorff_distance = []
Points = []
Polygons = []

f = np.loadtxt("/Users/pantelistassopoulos/Downloads/Line/Points_10/verticeszTreeLin
x = f[1:-1,0]
y = f[1:-1,1]
points = []

for i in range(len(x)):
    points.append([x[i], y[i]])

points = np.matrix(points)
Points.append(points)

plt.plot(points[:,0], points[:,1], 'r.', markersize = 1)

Polygon = []
for _ in range(len(points)-1):
    t = np.linspace(0,1,10)
    p1 = [points[_, 0], points[_, 1]]
    p2 = [points[_, 0], points[_+1, 1]]
    for _ in t:
        p = [float(p1[0])*(1-_) + float(p2[0])*_, float(p1[1])*(1-_) + float(p2[1])*
        Polygon.append(p)
Polygons.append(Polygon)
plt.plot(points[:,0], points[:,1], 'r-', markersize = 1)

for _ in range(10):
    f = np.loadtxt("/Users/pantelistassopoulos/Downloads/Line/Points_10/verticeszTre
    x = f[1:-1,0]
    y = f[1:-1,1]
    points = []

    for i in range(len(x)):
        points.append([x[i], y[i]])

    points = np.matrix(points)
    Points.append(points)

    hull = ConvexHull(points)

    plt.plot(points[:,0], points[:,1], 'r.', markersize = 2)

    r = random.random()

    b = random.random()

    g = random.random()

    color = (r, g, b)
```

```
    for simplex in hull.simplices:
        plt.plot(points[simplex,0], points[simplex,1], 'b-')

    Polygon = []
    for simplex in hull.simplices:
        t = np.linspace(0,1,10)
        p1 = [points[simplex[0], 0], points[simplex[0], 1]]
        p2 = [points[simplex[1], 0], points[simplex[1], 1]]
        for _ in t:
            p = [float(p1[0])*(1-_) + float(p2[0])*_, float(p1[1])*(1-_) + float(p2
            Polygon.append(p)
    Polygons.append(Polygon)



plt.show()
for j in range(1,len(Polygons)):

    Hausdorff_distance.append(directed_hausdorff(Points[0], Points[j])[0])

plt.plot(list(range(1,len(Polygons))), Hausdorff_distance)
Hausdorff_distance
```

Degree three vertex Tree Hausdorff distance

```
import numpy as np
import random
import numpy as np
from scipy.spatial import ConvexHull
from scipy.spatial.distance import directed_hausdorff
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import pandas


def func_powerlaw(x, m, c, c0):
    return c0 + x**m * c

def logistic(x, a, b, c, d):
    return float(a) / (1.0 + np.exp(-float(c) * (x - float(d)))) + float(b)


def rational(x):
    return x/(1+np.abs(x))

def rational2(x, a, b, c, d):
    return a*rational(b*(x-c))+d

#to contain file names
files = [0,1,2,3,4,5,6,7,8,9,10,11,12,13, 14]
Hausdorff_distance = []
Points = []
Polygons = []

file = open("/Users/pantelistassopoulos/Downloads/Line/ThreeVertex/combined.txt", "w

for _ in range(len(files)):
```

```
    f = np.loadtxt("/Users/pantelistassopoulos/Downloads/Line/ThreeVertex/verticeszT
    f1 = open("/Users/pantelistassopoulos/Downloads/Line/ThreeVertex/verticeszThreeV
    for line in f1:
            file.write(line)
    f1.close()
    x = f[1:-1,0]
    y = f[1:-1,1]
    points = []

    for i in range(len(x)):
        points.append([x[i], y[i]])

    points = np.matrix(points)

    hull = ConvexHull(points)

    plt.plot(points[:,0], points[:,1], 'r.', markersize = 1)

    r = random.random()

    b = random.random()

    g = random.random()

    color = (r, g, b)

    for simplex in hull.simplices:
        plt.plot(points[simplex,0], points[simplex,1], 'b-')

    Polygon = []
    for simplex in hull.simplices:
        t = np.linspace(0,1,10)
        p1 = [points[simplex[0], 0], points[simplex[0], 1]]
        p2 = [points[simplex[1], 0], points[simplex[1], 1]]
        for _ in t:
            p = [float(p1[0])*(1-_) + float(p2[0])*_, float(p1[1])*(1-_) + float(p2
            Polygon.append(p)
    Polygons.append(Polygon)
    Points.append(points)

circle = plt.Circle((0, 0), 2, color='g')
plt.gca().add_patch(circle)
plt.gca().set_aspect('equal', adjustable='box')

plt.show()

file.close()

for j in range(len(Points)):

    Hausdorff_distance.append(directed_hausdorff(Points[0], Points[j])[0])

#plt.plot(list(range(len(Points))), Hausdorff_distance, 'g+', markersize = 10)
print(Hausdorff_distance)

target_func = rational2
target_func2 = logistic
```

```
x = np.array(range(len(Points)))
y = np.array(Hausdorff_distance)
popt, pcov = curve_fit(target_func, x, y, maxfev = 2000)
X = np.linspace(0, len(Points))
#plt.plot(X, target_func(X, *popt), 'r--', label='rational', markersize = 1)
popt2, pcov2 = curve_fit(target_func2, x, y, maxfev = 2000)
#plt.plot(X, target_func2(X, *popt2), 'b--', label='logistic', markersize = 1)

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Hausdorff Distance Regression Three Vertex')
ax1.plot(X, target_func(X, *popt), 'r--', label='rational', markersize = 1)
ax1.plot(list(range(len(Points))), Hausdorff_distance, 'g+', markersize = 10)
ax2.plot(X, target_func2(X, *popt2), 'b--', label='logistic', markersize = 1)
ax2.plot(list(range(len(Points))), Hausdorff_distance, 'g+', markersize = 10)
ax1.legend()
ax2.legend()
popt
```

Tree insert edge left right

```
# add line segment left/ right

#Degree Three Vertex Tree Generation
def line(l, r):

    L = [1,2,3,4,5,6]
    L2 = [2,1,4,3,6,5]

    l_vertex = 3
    r_vertex = 5

    for _ in range(l):
        for  in range(len(L)):
            if L[ ] > l_vertex:
                L[ ] += 2
        for  in range(len(L2)):

            if L2[ ] > l_vertex:
                L2[ ] += 2
        L.append(l_vertex+1)
        L.append(l_vertex+2)
        L2.append(l_vertex+2)
        L2.append(l_vertex+1)
        l_vertex += 1
        r_vertex += 2


    for _ in range(r):
        for  in range(len(L)):
            if L[ ] > r_vertex:
                L[ ] += 2
        for  in range(len(L2)):

            if L2[ ] > r_vertex:
                L2[ ] += 2
```

```
        L.append(r_vertex+1)
        L.append(r_vertex+2)
        L2.append(r_vertex+2)
        L2.append(r_vertex+1)
        r_vertex += 1


    lines = []

    zipped_lists = zip(L, L2)

    sorted_pairs = sorted(zipped_lists)

    tuples = zip(*sorted_pairs)

    list1, list2 = [ list(tuple) for tuple in  tuples]
    for i in range(len(list1)):
        lines.append(str(list1[i])+"-"+str(list2[i]))

    return lines


def lines(N):
    for n in range(N):
        f = open('/Users/pantelistassopoulos/Downloads/Line/Points_50/TreeLine50_'+s

        Lines = line(n, N-n)

        for x in Lines:
            f.write(x)
            f.write('\n')
        f.close()

lines(50)
```

References