EE2703 Week 7

March 29, 2023

1 Libraries

[1]:
```
# Set up imports
%matplotlib ipympl
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from numpy import cos, sin, pi, exp
import random
import math
```

2 Part 1: Function

2.1 Simulated Annealing Function

[2]:
```
def SimAnn(f,r,t = 10, d = 0.99):
    # Initial temperature
    T = t
    decayrate = d
    # Set up some large value for the best cost found so far
    bestcost = 100000
    # Generate several values within a search 'space' and check whether the new␣
 ↪value is better
    # than the best seen so far.
    bestx = r[0]
    rl, rh = r[0], r[1]
    fig, ax = plt.subplots()
    xbase = np.linspace(rl,rh,100)
    ybase = f(xbase)
    ax.plot(xbase, ybase)
    xall, yall = [], []
    lnall,  = ax.plot([], [], 'ro')
    lngood, = ax.plot([], [], 'go', markersize=10)
    def onestep(frame):
        nonlocal bestcost, bestx, decayrate, T
        dx = (np.random.random_sample() - 0.5) * T
        x = bestx + dx
```

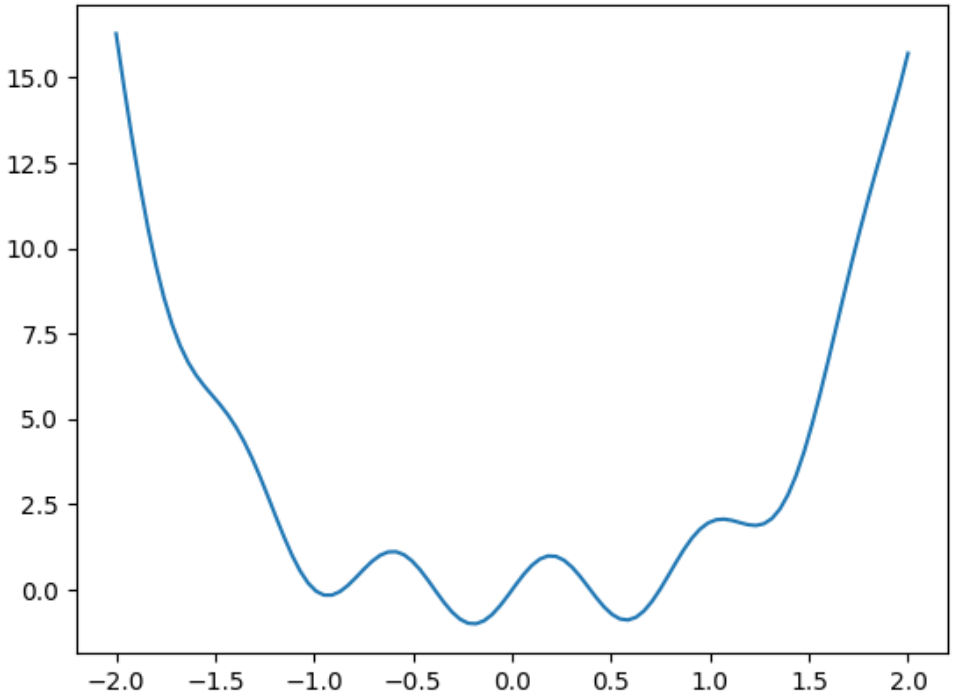
```
        y = f(x)
        if y < bestcost:
            bestcost = y
            bestx = x
            lngood.set_data(x, y)
        else:
            toss = np.random.random_sample()
            if toss < np.exp(-(y-bestcost)/T):
                bestcost = y
                bestx = x
                lngood.set_data(x, y)
            pass
        T = T * decayrate
        xall.append(x)

        yall.append(y)
        lnall.set_data(xall, yall)

    ani= FuncAnimation(fig, onestep, frames=range(10000), interval=10,␣
↪repeat=False)
    return ani
```

2.2 Function to test

[3]:
```
def f(x):
    return x**4 + np.sin(8*(x))
ans = SimAnn(f,[-2,2])
plt.show()
```



3 Part - 2: Travelling Salesman

The traveling salesman problem gives you a set of city locations (x, y coordinates). Your goal is to find a route from a given starting point that visits all the cities exactly once and then returns to the origin, with the minimum total distance covered (distance is measured as Euclidean distance $\sqrt{(x2-x1)^2+(y2-y1)^2}$. You will be given a file where the first line is the number of cities N, and the next N lines give the cities as a list of x, y coordinates: for example

```
4
0.0 1.5
2.3 6.1
4.2 1.3
2.1 4.5
```

Your goal is to give a sequence of numbers, for example [0, 3, 2, 1] which specifies the order in which to visit the cities. Note that after the last city you will come back to the first one in the list. Plot the cities with the path you have specified, and output the total length of the shortest path discovered so

[4]:
```
def Dataset(file_path):
    a = []
```

```
    with open(file_path, 'r') as file:
        data = file.readlines()
        data.pop(0)
    for lines in data:
        data = lines.split()
        b = [data[0],data[1]]
        a.append(b)
        npa = np.array(a,dtype = np.float64)
    return npa
```

[5]:
```
def distance(city1, city2):
    return math.sqrt((city1[0]-city2[0])**2 + (city1[1]-city2[1])**2)

def total_distance(path, cities):
    dist = 0
    for i in range(len(path)-1):
        dist += distance(cities[path[i]], cities[path[i+1]])
    dist += distance(cities[path[-1]], cities[path[0]])
    return dist
```

[6]:
```
def simulated_annealing(cities, T=1e8, alpha=1e-4, stopping_T=1e-8):
    cur_path = list(range(len(cities)))
    random.shuffle(cur_path)

    best_path = cur_path[:]
    best_distance = total_distance(best_path, cities)

    cur_distance = total_distance(cur_path, cities)

    while T > stopping_T:

        new_path = cur_path[:]
        rand_city1 = random.randint(0, len(new_path)-1)
        rand_city2 = random.randint(0, len(new_path)-1)
        new_path[rand_city1], new_path[rand_city2] = new_path[rand_city2],␣
 ↪new_path[rand_city1]

        new_distance = total_distance(new_path, cities)

        if new_distance < cur_distance:
            cur_path = new_path
            cur_distance = new_distance
            if cur_distance < best_distance:
                best_path = cur_path[:]
                best_distance = cur_distance
        else:
            delta = new_distance - cur_distance
```

```
            p = math.exp(-delta / T)
            if random.random() < p:
                cur_path = new_path
                cur_distance = new_distance

        T *= (1-alpha)

    return best_path, best_distance
```

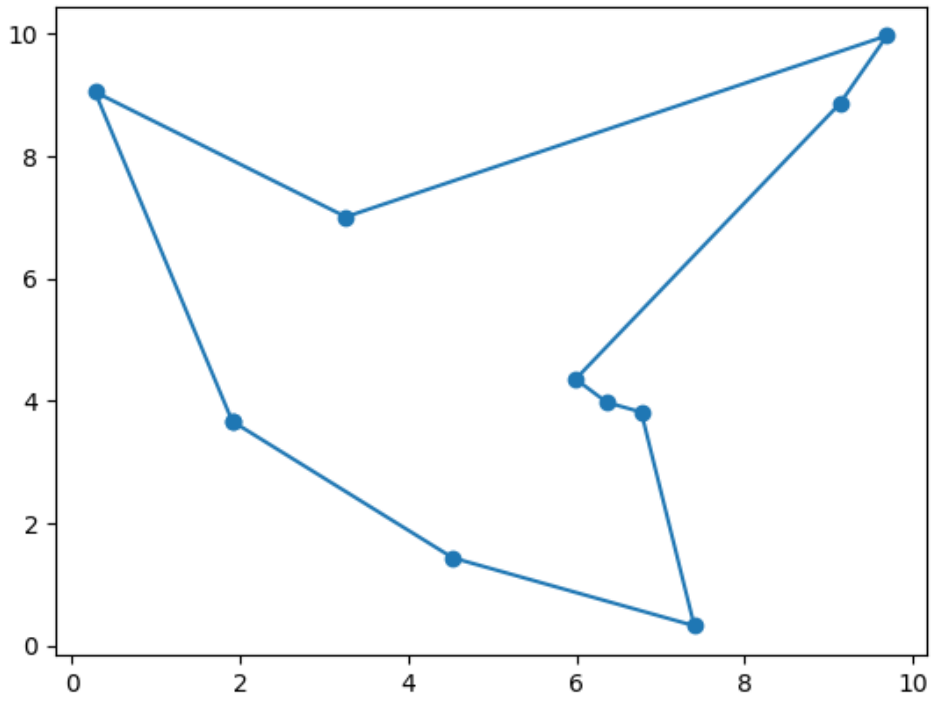
3.1 10 Cities

[7]:
```
cities = Dataset("tsp_10.txt")
path, best_distance = simulated_annealing(cities)
x_cities , y_cities = [],[]
for a in cities:
    x_cities.append(a[0])
    y_cities.append(a[1])
print(path)
print(best_distance)
```

```
[5, 6, 0, 2, 8, 9, 7, 1, 3, 4]
34.07656139463668
```

[8]:
```
x_cities , y_cities = [],[]
for a in cities:
    x_cities.append(a[0])
    y_cities.append(a[1])

x_cities = np.array(x_cities)
y_cities = np.array(y_cities)
xplot = x_cities[path]
yplot = y_cities[path]
xplot = np.append(xplot, xplot[0])
yplot = np.append(yplot, yplot[0])
plt.close()
plt.plot(xplot, yplot, 'o-')
plt.show()
```



3.2 100 Cities

[15]:
```
cities100 = Dataset("tsp_100.txt")
path100, best_distance100 = simulated_annealing(cities100 ,T=1e8, alpha=1e-4,␣
↪stopping_T=1e-8)
print(path100)
print(best_distance100)
```

```
[68, 51, 49, 13, 48, 75, 58, 46, 82, 33, 32, 77, 17, 88, 76, 6, 80, 36, 21, 54,
71, 90, 29, 59, 53, 97, 31, 55, 3, 22, 18, 7, 27, 14, 30, 86, 25, 19, 93, 65,
34, 0, 94, 74, 40, 8, 26, 23, 98, 62, 85, 10, 83, 20, 91, 28, 64, 73, 96, 11,
37, 9, 66, 5, 61, 63, 38, 57, 60, 24, 92, 72, 12, 95, 78, 67, 35, 69, 70, 1, 81,
44, 79, 43, 2, 89, 4, 41, 16, 99, 52, 39, 15, 50, 42, 84, 87, 45, 56, 47]
109.69714168568034
```

[16]:
```
x_cities100 , y_cities100 = [],[]
for a in cities100:
    x_cities100.append(a[0])
    y_cities100.append(a[1])
x_cities100 = np.array(x_cities100)
y_cities100 = np.array(y_cities100)
```

```
xplot = x_cities100[path100]
yplot = y_cities100[path100]
xplot = np.append(xplot, xplot[0])
yplot = np.append(yplot, yplot[0])

plt.close()
plt.plot(xplot, yplot, 'o-')
plt.show()
```