EE2703 - Week 7 : Annealing
Aayush Patel, EE21B003

March 30, 2023

1 Assignment

1.1 Part 1 - Simple

Write a function that takes as input the following:

- another function (note that in Python you can pass functions as arguments to other functions)
- starting point
- temperature
- learning/decay rate

and then proceeds to apply Simulated Annealing to minimize the function. This is more or less the same as what has been given above, but you need to encapsulate it all into a simple function and demonstrate how it works on arbitrary functions.

1.1.1 Annelaing

Annealing refers to a process of gradually cooling and solidifying a material to achieve a desired physical or chemical state. But in the context of optimization, annealing refers to a technique used to find the best solution to a problem by gradually decreasing the search space.
`Annelaing` mainly involves following 6 steps: -

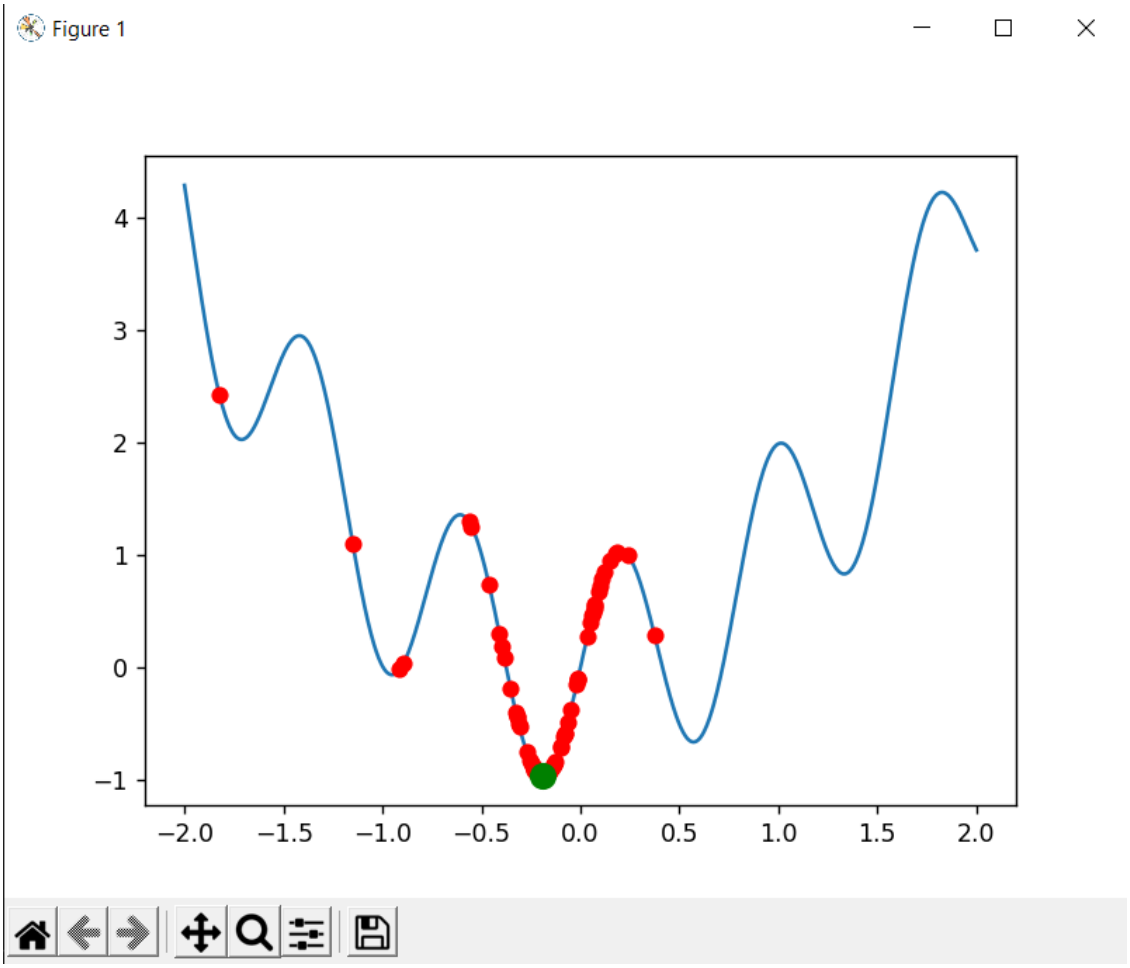
- `Initialization` : Select a staring point(starting solution).
- `Perturbation` : Make a small change in current solution.
- `Evaluation` : Calculate cost for the new solution
- `Acceptance` : If the current solution gives better result we accept it.
- `Cooling` : Decrease the Temperature Parameter.
- `Termination` : The process can't go infinitely long so we need to terminate after a particular number of steps.

By gradually reducing the temperature parameter, annealing allows the optimization algorithm to explore a wider range of the search space in the early stages, but then focuses on the most promising solutions as the search space is narrowed down.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def yfunc(x):
    return x**2 + np.sin(8*x)

def simulated_anealling(yfunc,start,temperature,decay_rate):
    # 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 = start
    rangemin, rangemax = -2, 2
    xbase=np.linspace(rangemin,rangemax,1000)
    ybase=yfunc(xbase)
    fig, ax = plt.subplots()
    ax.plot(xbase, ybase)
    xall, yall = [], []
    lnall,  = ax.plot([], [], 'ro')
    lngood, = ax.plot([], [], 'go', markersize=10)
    def onestep(frame):
        nonlocal bestcost, bestx, decay_rate, temperature
        # Generate a random value \in -2, +2
        dx = (np.random.random_sample() - 0.5) * temperature
        x = bestx + dx
        # print(f"Old x = {x}, delta = {dx}")
        y = yfunc(x)
        if y < bestcost:
            # print(f"Improved from {bestcost} at {bestx} to {y} at {x}")
            bestcost = y
            bestx = x
            lngood.set_data(x, y)
        else:
            toss = np.random.random_sample()
            if toss < np.exp(-(y-bestcost)/temperature):
                bestcost = y
                bestx = x
                lngood.set_data(x, y)
            # print(f"New cost {y} worse than best so far: {bestcost}")
            pass
        temperature = temperature * decay_rate
        xall.append(x)
        yall.append(y)
        lnall.set_data(xall, yall)
        # return lngood,
```

```
    ani= FuncAnimation(fig, onestep, frames=range(100), interval=100,␣
↪repeat=False)
    plt.show()

simulated_anealling(yfunc,-2,3,0.95)
```

The problem statement only said to take a function, starting point, temperature, cooling or decay rate. But to plot we need max and min range values. For the function used above I took the domain as [-2,2]. While evaluating for other functions the user has to change those value explicitely.



1.2 Part 2

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 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$).

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 far.

1.2.1 Part 2a

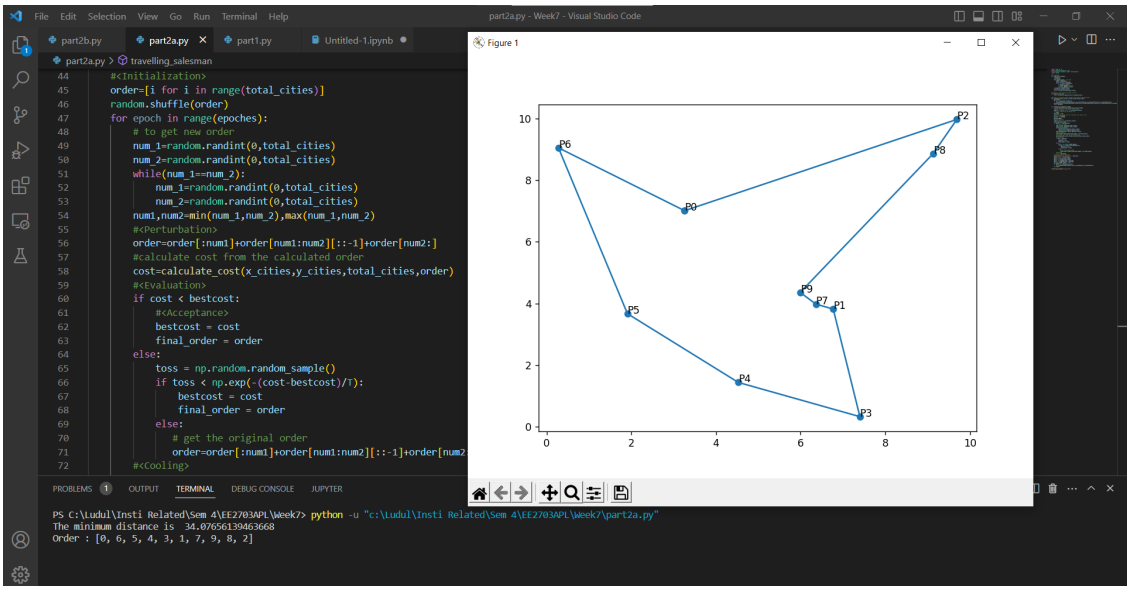
Get the minimized distance path for the file "tsp_10.txt"

Note that the difference in annealing in this problem is that we have a list of order as solution (in problem 1 we had just one value) So here, in the perturbation step I changed the current order by choosing a subarray of the given order and reversing it. Just randomly generating a new order won't work here because in perturbation step we have to change the current solution by a little.

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import random

#File Reading
def read_file(filename):
    x_cities=[]
    y_cities=[]
    with open(filename, 'r') as f:
        data=f.readlines()
        total_cities=int(data[0])
        for i in range(1,len(data)):
            x,y=data[i].split()
            x_cities.append(float(x))
            y_cities.append(float(y))
    x_cities=np.array(x_cities)
    y_cities=np.array(y_cities)
    return x_cities,y_cities,total_cities
```

```
#Function to give distance between 2 points
def dist(x1,y1,x2,y2):
    return np.sqrt(np.square(x2-x1)+np.square(y2-y1))

#Function to calculate total distance traversed by the given path
def calculate_cost(x_cities,y_cities,total_cities,order):
    distance=0
    for i in range(total_cities-1):
        ␣
↪distance+=dist(x_cities[order[i]],y_cities[order[i]],x_cities[order[i+1]],y_cities[order[i+1]
  ␣
↪distance+=dist(x_cities[order[total_cities-1]],y_cities[order[total_cities-1]],x_cities[order
    return distance

def travelling_salesman(filename):
    x_cities,y_cities,total_cities=read_file(filename)
    #labels to name all the points on the graph
    labels=['P'+str(i) for i in range(total_cities)]
    # Initial temperature and decay rate
    T = 100.0
    decayrate = 0.99
    # Set up some large value for the best cost found so far
    bestcost = 10000000
    #Number of iteraions
    epoches=10000
    #<Initialization>
    order=[i for i in range(total_cities)]
    random.shuffle(order)
    for epoch in range(epoches):
        # to get new order
        num_1=random.randint(0,total_cities)
        num_2=random.randint(0,total_cities)
        while(num_1==num_2):
            num_1=random.randint(0,total_cities)
            num_2=random.randint(0,total_cities)
        num1,num2=min(num_1,num_2),max(num_1,num_2)
        #<Perturbation>
        order=order[:num1]+order[num1:num2][::-1]+order[num2:]
        #calculate cost from the calculated order
        cost=calculate_cost(x_cities,y_cities,total_cities,order)
        #<Evaluation>
        if cost < bestcost:
            #<Acceptance>
            bestcost = cost
            final_order = order
        else:
            toss = np.random.random_sample()
```

```
            if toss < np.exp(-(cost-bestcost)/T):
                bestcost = cost
                final_order = order
            else:
                # get the original order
                order=order[:num1]+order[num1:num2][::-1]+order[num2:]
        #<Cooling>
        T = T * decayrate
    #<Termination> of annealing
    print("The minimum distance is ",bestcost)
    print("Order :",final_order)
    xplot = x_cities[final_order]
    yplot = y_cities[final_order]
    xplot = np.append(xplot, xplot[0])
    yplot = np.append(yplot, yplot[0])
    plt.plot(xplot, yplot, 'o-')
    for point in range(total_cities):
        plt.annotate(labels[point],xy=(x_cities[point],y_cities[point]))
    plt.show()

travelling_salesman("tsp_10.txt")
```

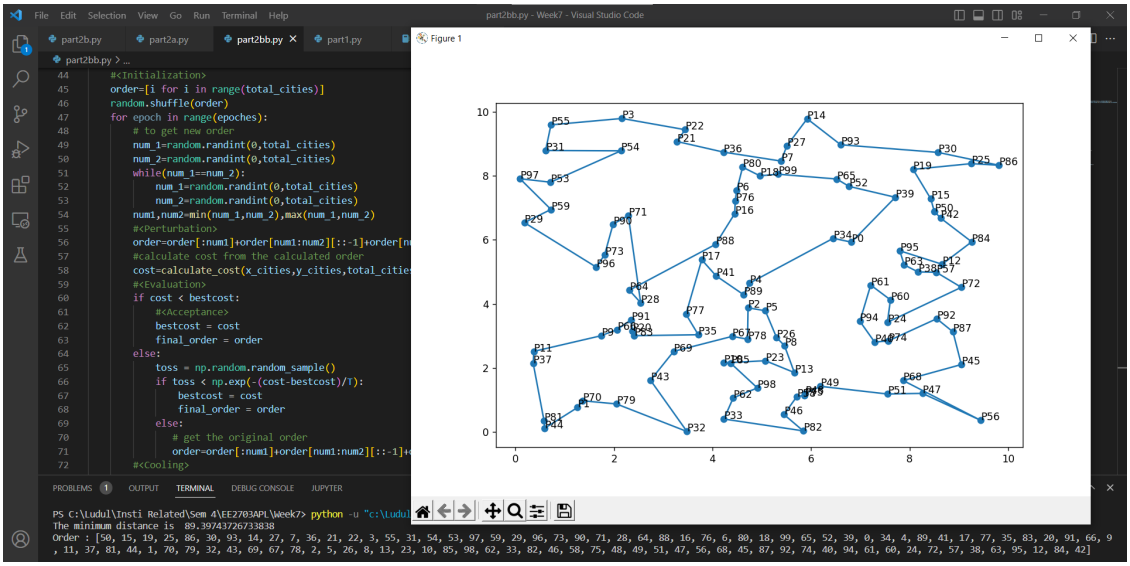


1.2.2 Part 2b

Get the minimized distance path for the file "tsp_100.txt".

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import random

#File Reading
def read_file(filename):
    x_cities=[]
    y_cities=[]
    with open(filename, 'r') as f:
        data=f.readlines()
        total_cities=int(data[0])
        for i in range(1,len(data)):
            x,y=data[i].split()
            x_cities.append(float(x))
            y_cities.append(float(y))
    x_cities=np.array(x_cities)
    y_cities=np.array(y_cities)
    return x_cities,y_cities,total_cities

#Function to give distance between 2 points
def dist(x1,y1,x2,y2):
    return np.sqrt(np.square(x2-x1)+np.square(y2-y1))

#Function to calculate total distance traversed by the given path
def calculate_cost(x_cities,y_cities,total_cities,order):
    distance=0
    for i in range(total_cities-1):
        ␣
 ↪distance+=dist(x_cities[order[i]],y_cities[order[i]],x_cities[order[i+1]],y_cities[order[i+1]
   ␣
 ↪distance+=dist(x_cities[order[total_cities-1]],y_cities[order[total_cities-1]],x_cities[order
    return distance

def travelling_salesman(filename):
    x_cities,y_cities,total_cities=read_file(filename)
    #labels to name all the points on the graph
    labels=['P'+str(i) for i in range(total_cities)]
    # Initial temperature and decay rate
    T = 100.0
    decayrate = 0.99
    # Set up some large value for the best cost found so far
    bestcost = 10000000
    #Number of iteraions
```

```
    epoches=10000
    #<Initialization>
    order=[i for i in range(total_cities)]
    random.shuffle(order)
    for epoch in range(epoches):
        # to get new order
        num_1=random.randint(0,total_cities)
        num_2=random.randint(0,total_cities)
        while(num_1==num_2):
            num_1=random.randint(0,total_cities)
            num_2=random.randint(0,total_cities)
        num1,num2=min(num_1,num_2),max(num_1,num_2)
        #<Perturbation>
        order=order[:num1]+order[num1:num2][::-1]+order[num2:]
        #calculate cost from the calculated order
        cost=calculate_cost(x_cities,y_cities,total_cities,order)
        #<Evaluation>
        if cost < bestcost:
            #<Acceptance>
            bestcost = cost
            final_order = order
        else:
            toss = np.random.random_sample()
            if toss < np.exp(-(cost-bestcost)/T):
                bestcost = cost
                final_order = order
            else:
               # get the original order
               order=order[:num1]+order[num1:num2][::-1]+order[num2:]
        #<Cooling>
        T = T * decayrate
    #<Termination> of annealing
    print("The minimum distance is ",bestcost)
    print("Order :",final_order)
    xplot = x_cities[final_order]
    yplot = y_cities[final_order]
    xplot = np.append(xplot, xplot[0])
    yplot = np.append(yplot, yplot[0])
    plt.plot(xplot, yplot, 'o-')
    for point in range(total_cities):
        plt.annotate(labels[point],xy=(x_cities[point],y_cities[point]))
    plt.show()

travelling_salesman("tsp_100.txt")
```



Note: I have used a jupyter notebook just to make the pdf easily. I am not attaching the jupter notebook as it gives some problems while plotting and animating. I am attaching 3 individual run-alone pyhton files.