EE2703: Simulated Annealing
Programming Assignment # 7

Student's Name, Roll Number :
Annangi Shashank Babu, EE21B021

March 30, 2023

Contents

List of Figures

List of Tables

1 Function Minimisation

Simulated Annealing is a popular **optimization** algorithm used to find the global minimum of a given function. The algorithm uses the concepts of decaying learning rate and randomisations to find the optimal solution.

Simulated Annealing works by deviating from the current point by a random distance (but as temperature decreases we should deviate less from the current point) and evaluating the function at that point.The new point becomes the current point if it has a lower function value than the current point. Based on the temperature and learning rate, the algorithm may still accept the new point with a given probability even if it has a greater function value than the existing point. The likelihood of accepting a new point with a higher function value reduces as the temperature rises. This procedure continues until the algorithm reaches a maximum number of iterations.

```
def mysimulated_anneling(func,start,temperature,decayrate):
    """function to implement simulated anneling

    :func: the function that has to be minimised
    :start: starting point
    :temperature:
    :decayrate:
    :returns: the minima

    """
    bestcost = 1e9
    bestx = start
    xbase = np.linspace(-2, 2, 100)
    ybase = func(xbase)
    rangemin, rangemax = -2, 2
    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, decayrate,temperature
        # Generate a random value \in -2, +2
        dx = (np.random.random_sample() - 0.5) *temperature
        x = bestx +dx
        y = func(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)/temperature):
                bestcost = y
                bestx = x
                lngood.set_data(x, y)

        temperature = temperature * decayrate
        xall.append(x)
        yall.append(y)
        lnall.set_data(xall, yall)

    ani= FuncAnimation(fig, onestep, frames=range(100), interval=100, repeat=False)
    plt.show()
    #for better accuracy
```

```
    for i in range(10000) :
        onestep(i)
    return bestx
```

1.1 Ouptut

for the function : $f(x) = x^2 + sin(8 * x)$

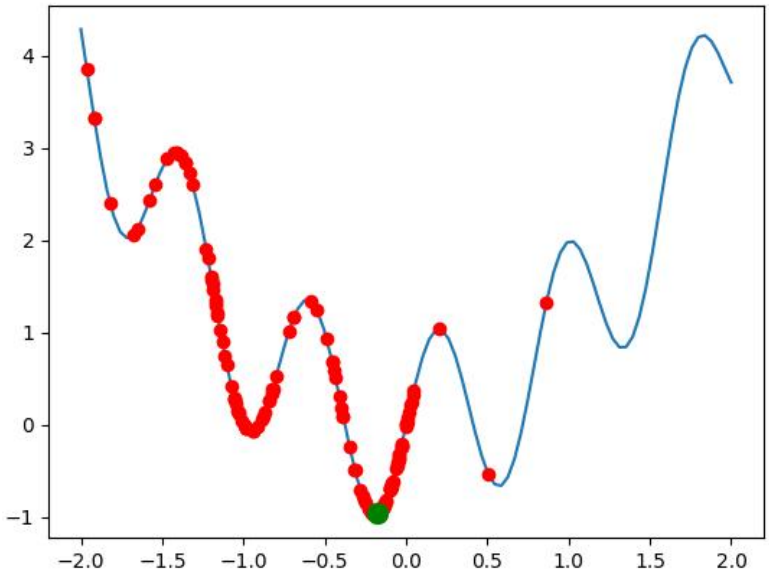

Figure 1: Output for the given function

Output of the code : The minimum value occurs at -0.18099753460055526, minimum being -0.9597074766558362

2 Travelling Salesman Problem

2.1 Recursion

The first brute force approach to solve the Travelling salseman problem is to calculate the distance covered for every single path and findout the minimum travel path, this can be achived from recursion which is of factorial time complexity, with introduction of dynamic programming we can reduce to time complexity to exponential.

```
def recurse(visited,coordinates,currcity = 0):
    """Recurse to find the solution to travelling salesman problem

    """
    order = []
    order.append(currcity)
    if visited.all() == 1:
        return distance(coordinates[currcity],coordinates[0]),order
    ans = 1e100
    for i in range(len(visited)):
        if visited[i] == 0 :
            visited_updated = np.copy(visited)
```

```
            visited_updated[i] = 1
            ansnew, ordernew = recurse(visited_updated, coordinates, i)
            ordernew = np.copy(ordernew)
            ansnew += distance(coordinates[i], coordinates[currcity])
            if ansnew < ans:
                ans = ansnew
                ordermin = np.copy(ordernew)
    order = np.append(order, ordermin)
    return ans, order
```

2.1.1 Output

The output for the case of 10 cities : the order and the total distace of travel are mentioned

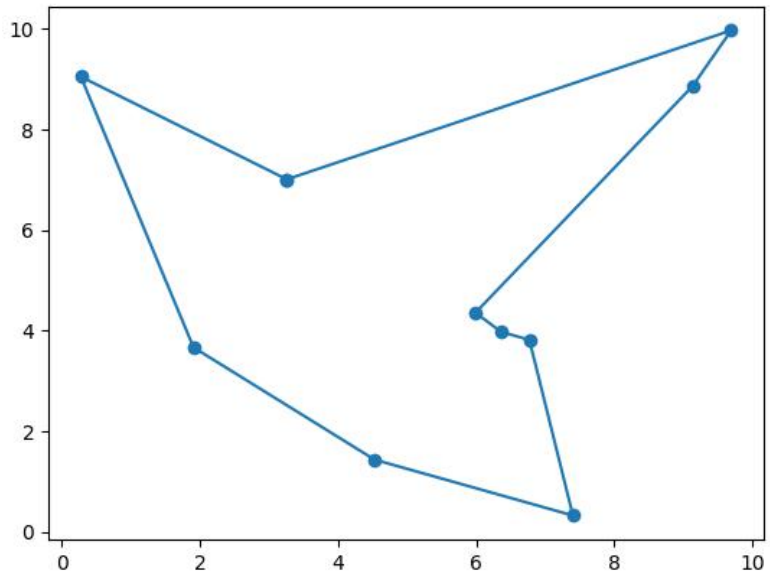

Figure 2: Ouput using recursion

in the file named **r out10.txt**.

2.2 Simulated Annealing

Now we use the simulated annealing algorithm to solve the travelling salesman problem, the optimisation algoritms come into play when the time complexity of solving a problem is exponential or factorial and definite solution for large inputs cant be found in finite time by a modern computer

- Choose a random permutation of the cities to visit.
- Initially, the temperature is high, allowing the algorithm to accept worse solutions. As the temperature decreases, the algorithm becomes more selective and only accepts better solutions.
- If the new solution is better than the current solution, accept it as the new current solution. If the new solution is worse than the current solution, accept it with a certain

probability that depends on the temperature and the difference in cost between the new and current solutions.

- After the iterations, return the best solution found, which is the solution with the lowest cost.

```
def mysimulated_anneling(visited,coordinates,temperature,starting_order,decayrate):
    """using simulated anneling to solve tsp

    :visited: the array that keeps track of visited cities
    :coordinates: coordinates of the cities
    :starting_order: the starting plan of travel
    """
    global bestcost,bestorder
    random_order = np.copy(starting_order)
    random_order = shuffle(random_order)
    newcost = cost(random_order)
    if bestcost > newcost:
        bestcost = newcost
        bestorder = np.copy(random_order)
    else :
        toss = np.random.random_sample()
        if toss < np.exp(-(newcost-bestcost)/temperature) :
            bestcost = newcost
            bestorder = np.copy(random_order)
    temperature = decayrate * temperature
    return temperature
```

2.2.1 Output for 10 cities

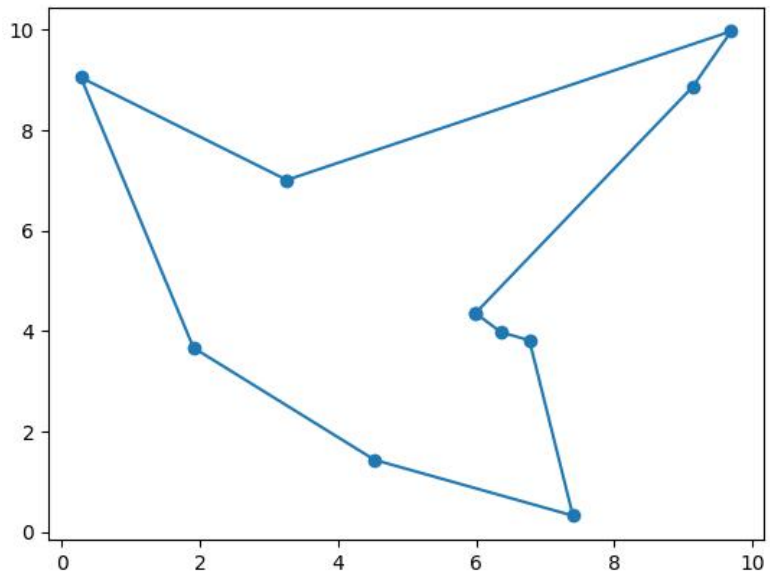

Figure 3: The output for the 10 cities testcase

the order and the total distace of travel are mentioned in the file named **out10.txt**.

2.2.2 Output for 100 cities

For the 100 cities case :

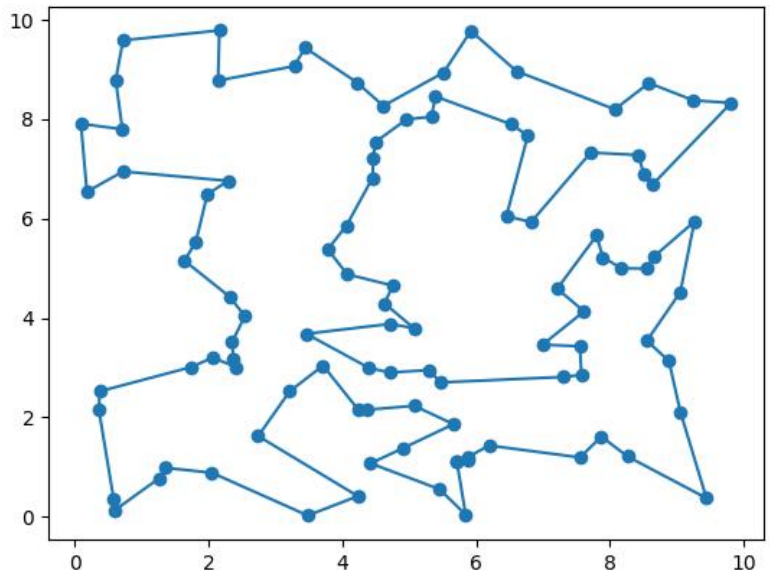

Figure 4: The output for the 100 cities testcase

the order and the total distace of travel are mentioned in the file named **out100.txt**.

3 Conclution

Both parts require a solid understanding of optimization techniques, such as Simulated Annealing and Reccursion, as well as problem-solving skills to apply these techniques to the given problems. The ability to read and parse input files and generate output plots is also necessary.

The hyper-parameters must be tuned properly to get better results, such as selecting a decay rate so that at the end of all the iterations the temperature is relatively small quantity (around -1e-5 was giving perfect result for above case) and decay rate must not be too small also which will lead to a rapid decay in temperature and causes to end up in a local minima.

Overall, this assignment tests our knowledge and skills in optimization and problem-solving.