Applied Programming Lab - week 7

Nachiket Mahesh Dighe EE21B093

March 29, 2023

1 Importing Libraries

[1]:
```
# Set up imports
%matplotlib ipympl
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import random
```

2 Part-1 : Apply Simulated Annealing

2.1 1 - Variable Function

[46]:
```
# Function with many minima
def yfunc(x):
    return x**2 + np.sin(8*x) #same example as given by sir

xbase = np.linspace(-2, 2, 10000)
ybase = yfunc(xbase)
```

Generating some function which has multiple minimas

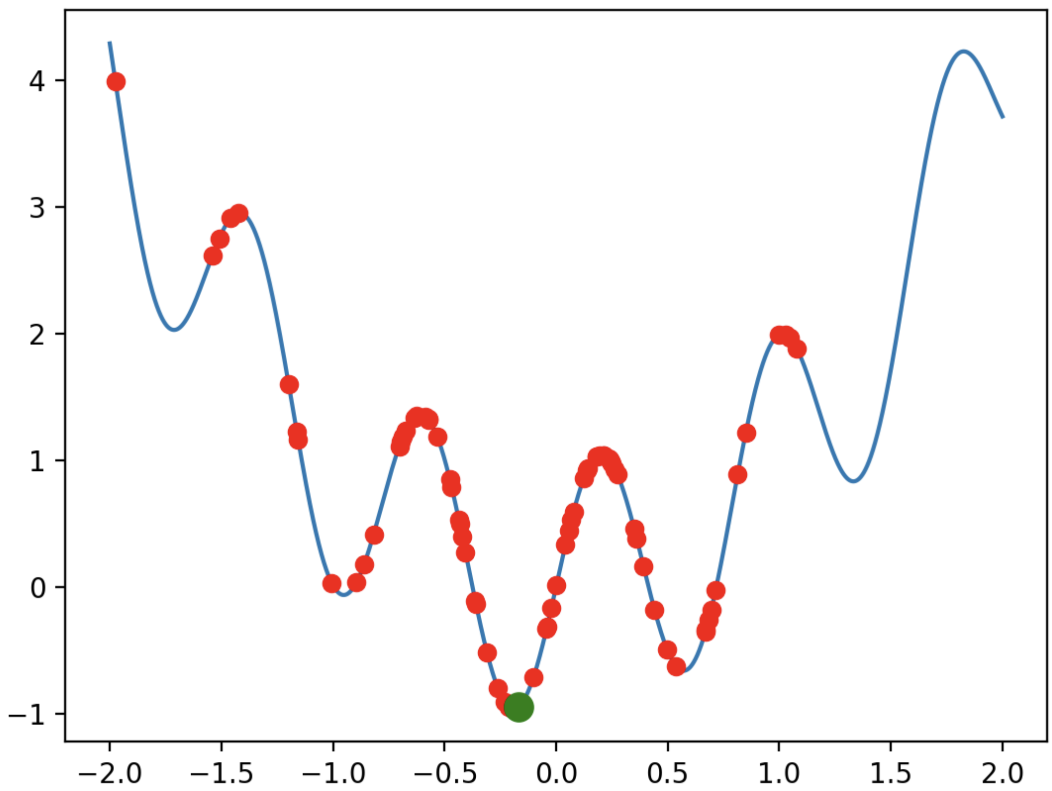
2.2 General Function:

[47]:
```
sp = -2 #starting point
T = 3 #Initial Temperature
bestcost = 100000
def gen_func(f , sp , bc , T , dr = 0.95):
    dx = (np.random.random_sample() - 0.5) * T  # generate a random value from␣
↪xmin to xmax
    x = sp + dx #shift the value of x from starting point randomnly
    y = f(x)
    if y < bc: #check if new value is less than bestcost
        bc = y
        sp = x

    else:
        toss = np.random.random_sample() #generate a point from 0 to 1
```

```
        if toss < np.exp(-(y-bc)/T): #If P is high then channge the values
            bc = y
            sp = x
        pass
    T = T * dr #decrease the temperature
    return x , y , sp , bc , T
```

- In this Code , gen_func() is a function which takes in a function as an argument. It also takes in the starting point , bestcost , Temperature and the decayrate.
- In this function we start off at some initial point . Then , we randomnly move to a different point annd check the value of the function(f) at that point . If it is less than the bestcost which we initialized then , we consider that point as the starting point for our next iteration.
- If the value is not small then we check for the Probability to move the point by comparing it with a random toss generated .
- Then for each iteration we change the starting point and also the Temperature as `T = T*(decay_rate)` .
- Finally after multiple iterations , we get the optimal minima of a particular Function.

[48]:
```
fig1, ax1 = plt.subplots()
ax1.plot(xbase, ybase)
xall, yall = [], []
lnall,  = ax1.plot([], [], 'ro')
lngood, = ax1.plot([], [], 'go', markersize=10)

def onestep(frame):
    global sp , bestcost , T #to ensure that after each function call the values␣
↪change
    x , y , sp , bestcost , T = gen_func(yfunc , sp ,bestcost, T)
    lngood.set_data(x, y)
    xall.append(x)
    yall.append(y)
    lnall.set_data(xall, yall)
    return x , y
ani= FuncAnimation(fig1, onestep, frames=range(100), interval=100, repeat=False)
plt.show()
```



- As seen from the above graph , we start from a point and then by using FuncAnimation the initial point gets shifted.
- This is done by the variable `frames` which iterates `100` times . For each iteration , the point gets shifted and so the animation of that gets displayed .
- Finally, after `100` iterations the final point reaches the optimal minima of the Function

3 Part2 - Travelling Salesman

[49]:
```
cities = 0
def readfile(lines):
    global cities
    city_points = []
    for line in lines:
        items = line.strip().split(" ")
        if len(items) == 1:
            cities = int(items[0])

        else:
```

```
            city_points.append([float(items[0]) , float(items[1])])

    return city_points
textfile = open('tsp_100.txt' , 'r')
city_points = readfile(textfile)
print(cities)
print(city_points)
```

```
100
[[6.82, 5.93], [1.26, 0.77], [4.72, 3.88], [2.16, 9.79], [4.75, 4.65], [5.07,
3.79], [4.49, 7.54], [5.39, 8.46], [5.46, 2.7], [1.74, 3.01], [4.23, 2.16],
[0.37, 2.52], [8.66, 5.24], [5.66, 1.86], [5.92, 9.78], [8.43, 7.28], [4.45,
6.82], [3.78, 5.38], [4.96, 8.0], [8.07, 8.2], [2.37, 3.16], [3.28, 9.07],
[3.44, 9.44], [5.07, 2.23], [7.55, 3.43], [9.25, 8.38], [5.3, 2.95], [5.51,
8.94], [2.54, 4.04], [0.18, 6.54], [8.58, 8.73], [0.61, 8.79], [3.48, 0.02],
[4.23, 0.41], [6.45, 6.05], [3.71, 3.04], [4.22, 8.74], [0.36, 2.14], [8.17,
5.0], [7.71, 7.33], [7.29, 2.81], [4.07, 4.88], [8.63, 6.68], [2.74, 1.62],
[0.59, 0.12], [9.05, 2.11], [5.45, 0.55], [8.26, 1.21], [5.88, 1.2], [6.19,
1.43], [8.5, 6.89], [7.55, 1.19], [6.77, 7.67], [0.71, 7.8], [2.14, 8.78],
[0.72, 9.59], [9.44, 0.37], [8.54, 4.99], [5.71, 1.11], [0.72, 6.95], [7.61,
4.13], [7.21, 4.59], [4.42, 1.07], [7.89, 5.22], [2.31, 4.43], [6.52, 7.9],
[2.06, 3.2], [4.4, 2.99], [7.87, 1.61], [3.21, 2.52], [1.36, 0.98], [2.29,
6.76], [9.05, 4.52], [1.81, 5.54], [7.57, 2.85], [5.87, 1.15], [4.46, 7.22],
[3.46, 3.68], [4.71, 2.9], [2.05, 0.88], [4.61, 8.27], [0.58, 0.35], [5.84,
0.04], [2.41, 3.01], [9.26, 5.94], [4.37, 2.15], [9.81, 8.33], [8.88, 3.13],
[4.06, 5.86], [4.63, 4.29], [1.98, 6.49], [2.35, 3.51], [8.55, 3.54], [6.6,
8.97], [6.99, 3.46], [7.8, 5.66], [1.64, 5.15], [0.09, 7.91], [4.91, 1.38],
[5.33, 8.05]]
```

- We first extract the inputs from the file given to us.
- Then we store the inputs to a variable .
- In this case, the city coordinates are stored in a 2D list named `city_points` and the total number of cities is stored in the variable `cities`.

[50]:
```
def distance(city1 , city2):
    x1 , y1 = city1
    x2 , y2 = city2
    return ((x1-x2)**2 + (y1-y2)**2)**0.5

def total_dist(city):
    sum = 0
    n = len(city)
    for i in range(n-1):
        sum += distance(city[i] , city[i+1])
    sum += distance(city[0] , city[n-1]) #connecting the start and end points
    return sum

print(total_dist(city_points))
```

```
500.7984267365113
```

- In this code all we are doing is taking two cities coordinates and finding the distance between them using the `distance()` function .
- Making use of this function , we find the total distance taken by the salesman to complete his Journey using the `total_dist` function which takes in the coordinates of the cities .

[51]:
```python
initial_route = [x for x in range(0,cities)]
random.shuffle(initial_route) #generate a random route
city = []
for i in initial_route:
    city.append(city_points[i])

T = 5
bestdist = total_dist(city) #total distance for that given route
```

- In this code, we first take some random starting point .
- In this case , the starting point is some route the salesman uses . This is stored in the variable `initial_route`
- Using this route , we find the total distance which is nothing but `bestcost` in this case and is stored in the variable `bestdist`.
- I have initialized the Temperature in this case to `5`.

3.1 Annealing():

[52]:
```python
def annealing(city , initial_route , bestdist , T , dr = 0.95):
    new_route = initial_route
    index1 = random.randint(0, cities - 1)
    index2 = random.randint(0, cities - 1)
    new_route[index1], new_route[index2] = new_route[index2], new_route[index1]
    new_city = []
    for i in new_route:
        new_city.append(city[i])

    new_dist = total_dist(new_city)
    if new_dist < bestdist:
        initial_route = new_route
        bestdist = new_dist
        city = new_city

    else:
        toss = np.random.random_sample()
        if toss < np.exp(-(new_dist-bestdist)/T):
            initial_route = new_route
            bestdist = new_dist
            city = new_city
        pass
    T = T*dr
```

```
    return city , initial_route , bestdist , T
```

- This Function takes the city coordinates , initial route , initial distance , Temperature and decay rate as arguments.
- In this Function , we make a new route by swapping the coordinates of two random cities .
- For this New Route , we find the total distance of that new route .
- We then check the initial distance and the new distance . If it is less than the initial distance which we initialized then we consider this to be our starting route.
- If the value is not small then we check for the Probability to still change the route by comparing it with a random toss generated .
- Finally after multiple iterations , we get the optimal minimum distance needed by the Salesman to travel.

[53]:
```
for i in range(100000):
    city , initial_route , bestdist , T = annealing(city , initial_route ,␣
 ↪bestdist , T)
final_route = initial_route
final_city = city
print(f"Minimum distance needed to travel : {bestdist}")
print("The Path needed to achieve this Distance is: ")
print(final_route)
```

```
Minimum distance needed to travel : 416.0295287123271
The Path needed to achieve this Distance is:
[63, 77, 78, 17, 85, 22, 40, 98, 20, 96, 83, 33, 62, 68, 47, 54, 51, 30, 99, 41,
75, 12, 56, 49, 74, 16, 73, 71, 46, 29, 72, 24, 64, 80, 37, 76, 79, 45, 10, 21,
48, 67, 95, 97, 90, 14, 82, 50, 3, 35, 19, 66, 81, 5, 34, 31, 7, 88, 55, 11, 27,
53, 28, 92, 4, 13, 93, 87, 23, 86, 9, 0, 52, 8, 91, 25, 84, 18, 89, 44, 58, 42,
32, 38, 39, 43, 70, 26, 57, 61, 36, 94, 6, 65, 1, 59, 15, 2, 60, 69]
```

- All this code does is it iterates some `n` times which in this case is `100000` .
- More the number of iterations more close we get to the optimal solution.
- As seen above the minimum distance needed to travel by the salesman is given and the order in which he needs to visit the cities is also given.

[54]:
```
# Example Python input and plot
fig2, ax2 = plt.subplots()
xcities = []
ycities = []
for i in range(cities):
    xcities.append(final_city[i][0])
    ycities.append(final_city[i][1])

xcities = np.array(xcities)
ycities = np.array(ycities)
```

```
# Rearrange for plotting
xplot = xcities
yplot = ycities
xplot = np.append(xplot, xplot[0])
yplot = np.append(yplot, yplot[0])
ax2.plot(xplot, yplot , 'o-')
plt.show()
```