EE 511 SIMULATION METHODS FOR STOCHASTIC SYSTEMS

PROJECT – 5

ABINAYA MANIMARAN

SPRING 2018

05/04/2018

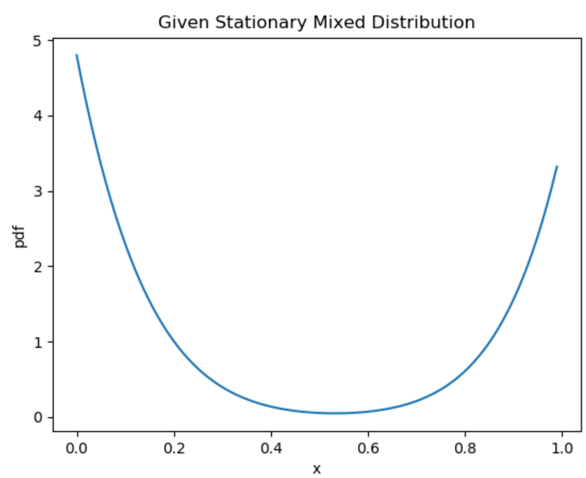
QUESTION 1 – MCMC FOR SAMPLING

Markov Chain Monte Carlo Simulation for sampling from given distribution:
Metropolis Hastings Algorithm:

- Generate Initial sample from the given distribution
- Choose a proposal pdf. It should be a symmetric pdf
- Generate the next sample given the previous sample from the proposal pdf
  - Meaning, the previous sample will be the mean if the proposal pdf is Gaussian
- Calculate the acceptance probability based on the new and previous sample
- If a random number generated is less than or equal to the acceptance probability, then accept the sample and append to the accepted list

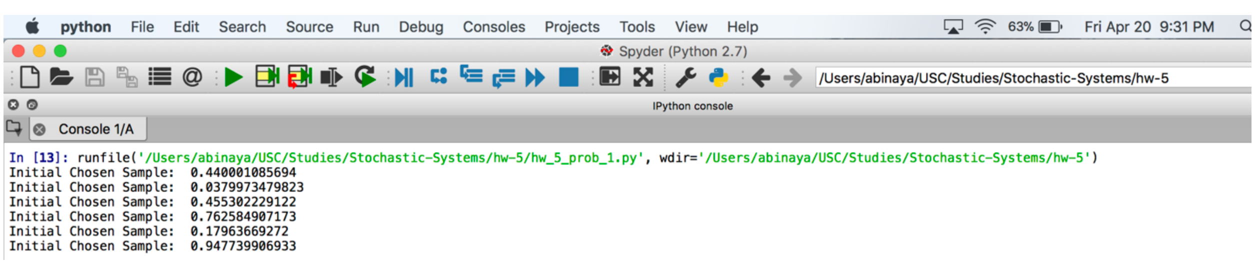
Given Distribution: 60% in Beta(1,8) and 40% in Beta(9,1) distribution

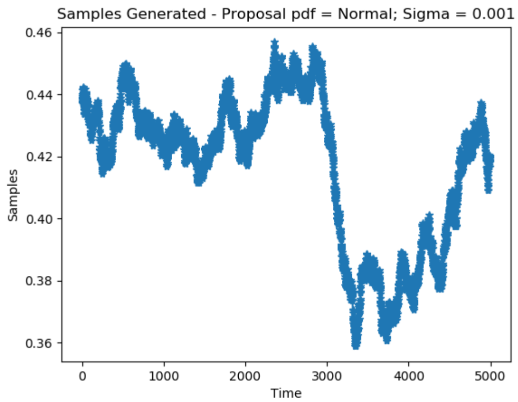
The pdf of the Function looks like this:

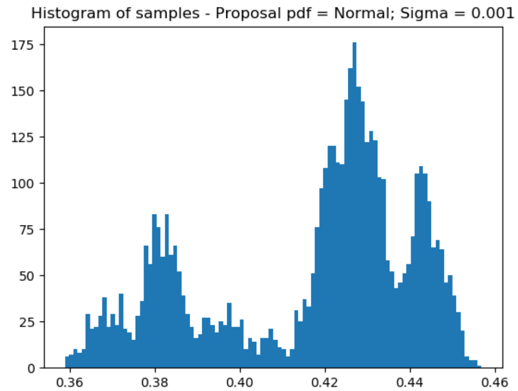


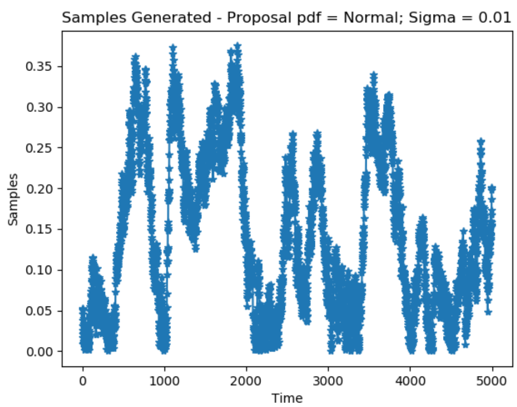
Metropolis Hastings Algorithm implemented for various parameters:

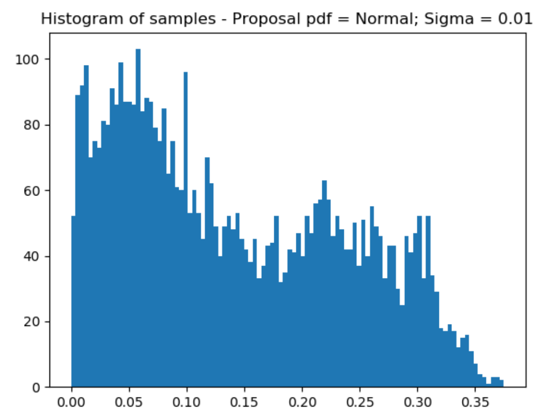
For Normal Distribution with different Sigma Values:

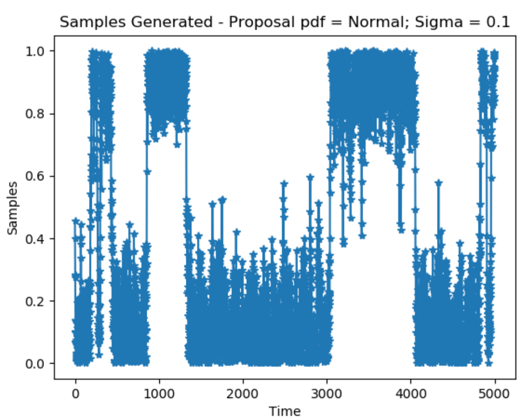


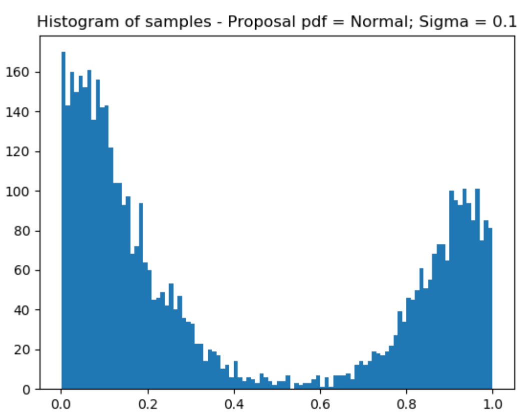

Samples Generated - Proposal pdf = Normal; Sigma = 0.001
Samples
Time

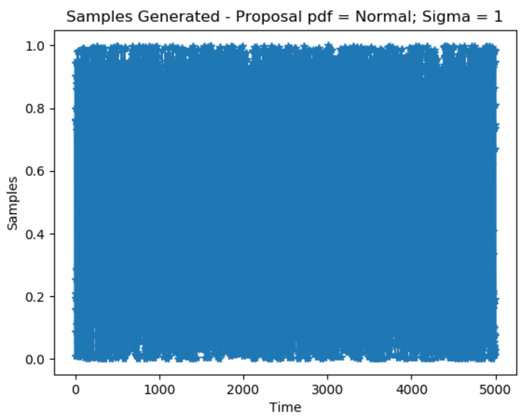

Histogram of samples - Proposal pdf = Normal; Sigma = 0.001

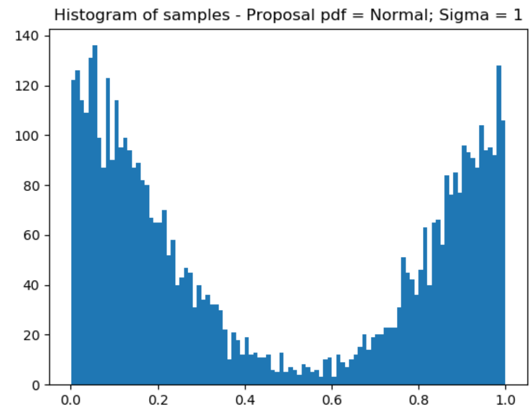

Samples Generated - Proposal pdf = Normal; Sigma = 0.01
Samples
Time

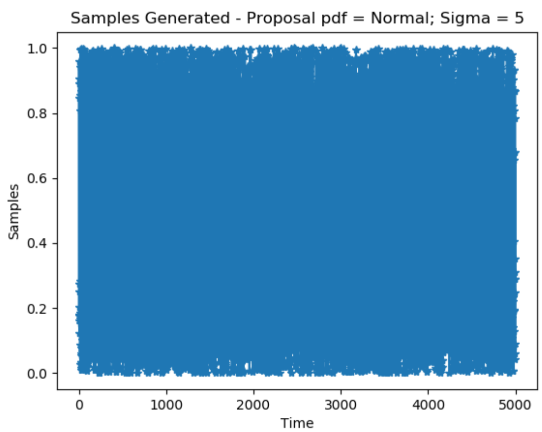

Histogram of samples - Proposal pdf = Normal; Sigma = 0.01

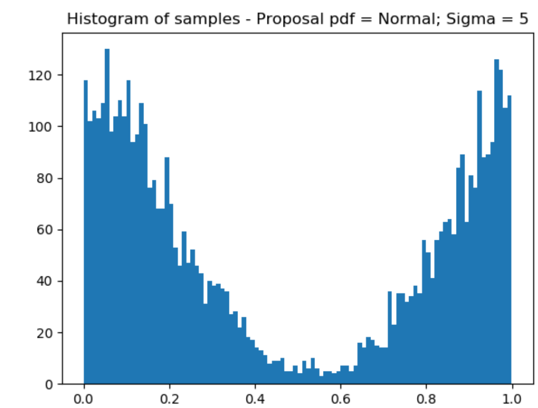

Samples Generated - Proposal pdf = Normal; Sigma = 0.1
Samples
Time

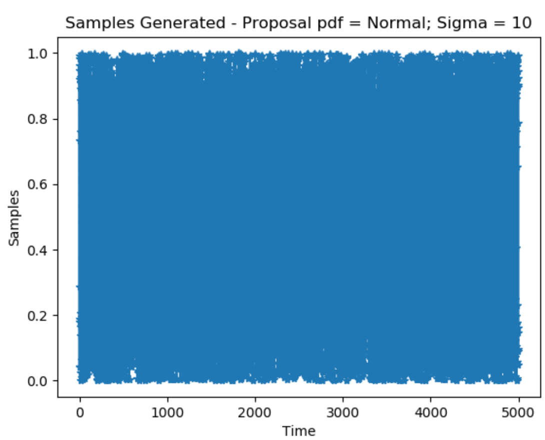

Histogram of samples - Proposal pdf = Normal; Sigma = 0.1

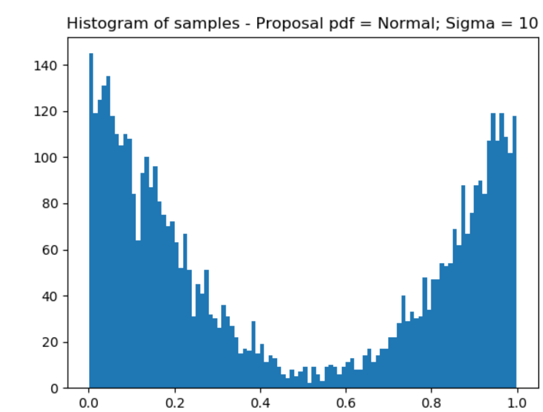

Samples Generated - Proposal pdf = Normal; Sigma = 1
Samples
Time
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000


Histogram of samples - Proposal pdf = Normal; Sigma = 1
0
20
40
60
80
100
120
140
0.0
0.2
0.4
0.6
0.8
1.0

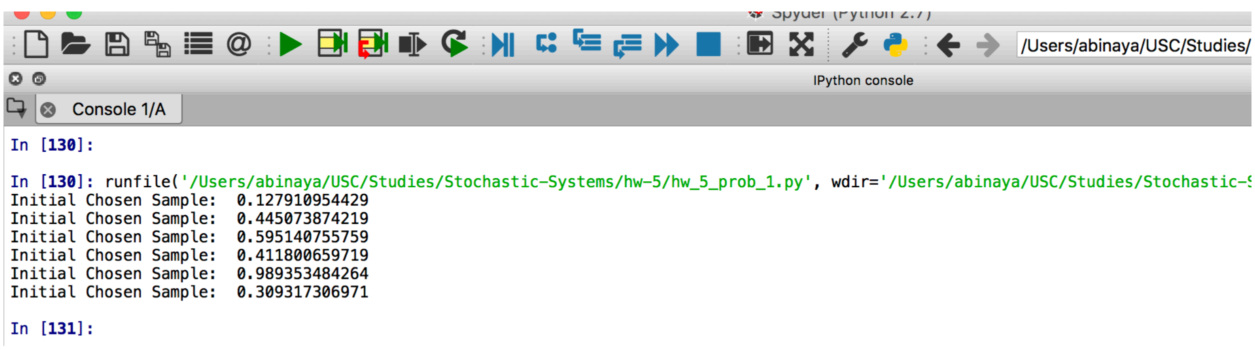

Samples Generated - Proposal pdf = Normal; Sigma = 5
Samples
Time
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000

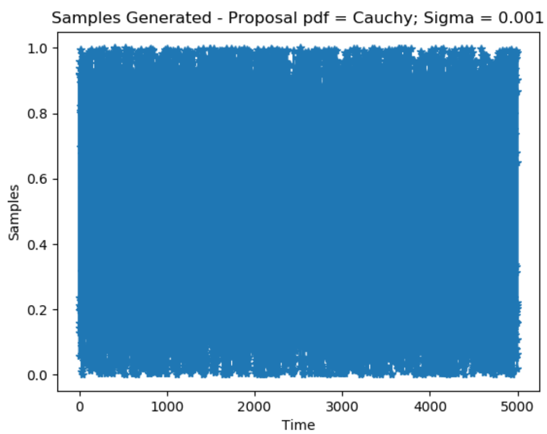

Histogram of samples - Proposal pdf = Normal; Sigma = 5
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0

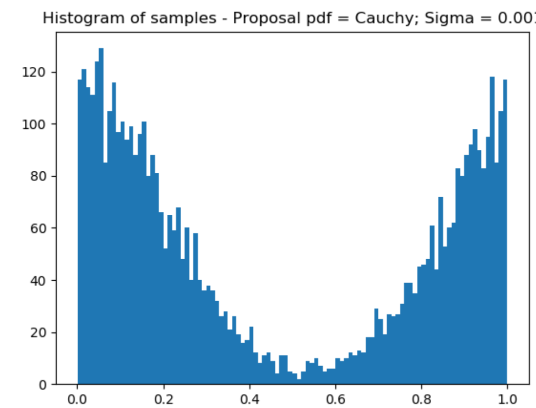

Samples Generated - Proposal pdf = Normal; Sigma = 10
Samples
Time
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000

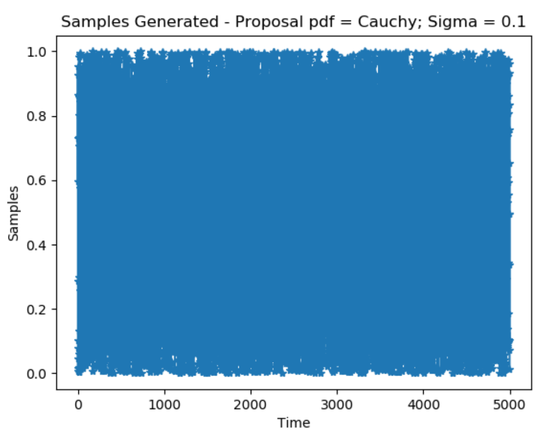

Histogram of samples - Proposal pdf = Normal; Sigma = 10
0
20
40
60
80
100
120
140
0.0
0.2
0.4
0.6
0.8
1.0

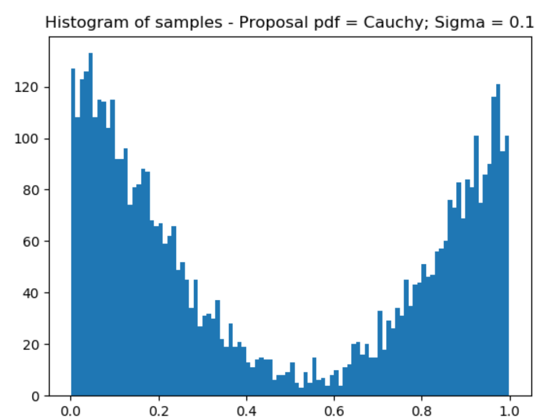
For Cauchy Distribution with different Sigma Values:











Discussion:

- All the above results show that, both Normal and Cauchy distribution tried with different variances for the Mixed Distribution
- Lower the variance of the proposal pdf distribution, even if numerous number of samples were generated, the distribution did not converge.
- This can be seen clearly from the Normal Distribution pdf chosen. As the variance was lower, the next sample generated at every iteration was very close the previous sample. This made the distribution not converge to the given stationary mixture distribution
- And various initial points were chosen for this process to see if they converge
- The theoretical distribution was given in the above section. The theoretical distribution looks the same as the distributions of the generated samples
- This obviously shows that Metropolis Hastings algorithm is one of the easiest way to generate samples from a mixed distribution.

QUESTION 2– MCMC FOR OPTIMIZATION

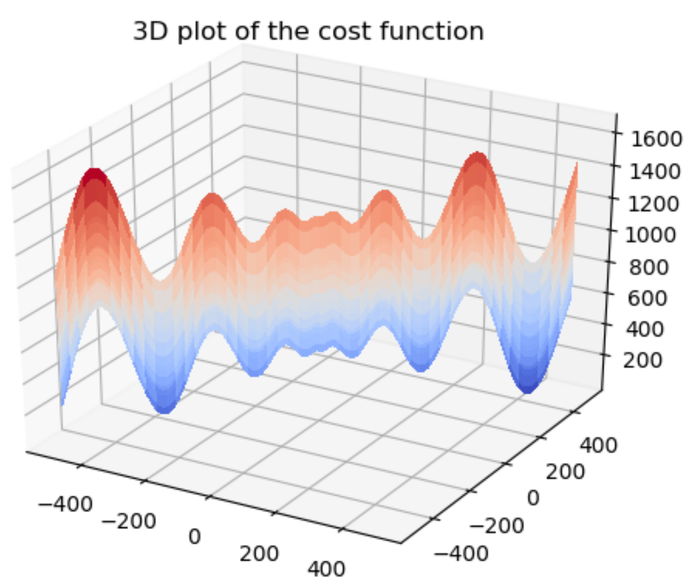
Markov Chain Monte Carlo Simulation for Optimization:
Simulated Annealing Algorithm:

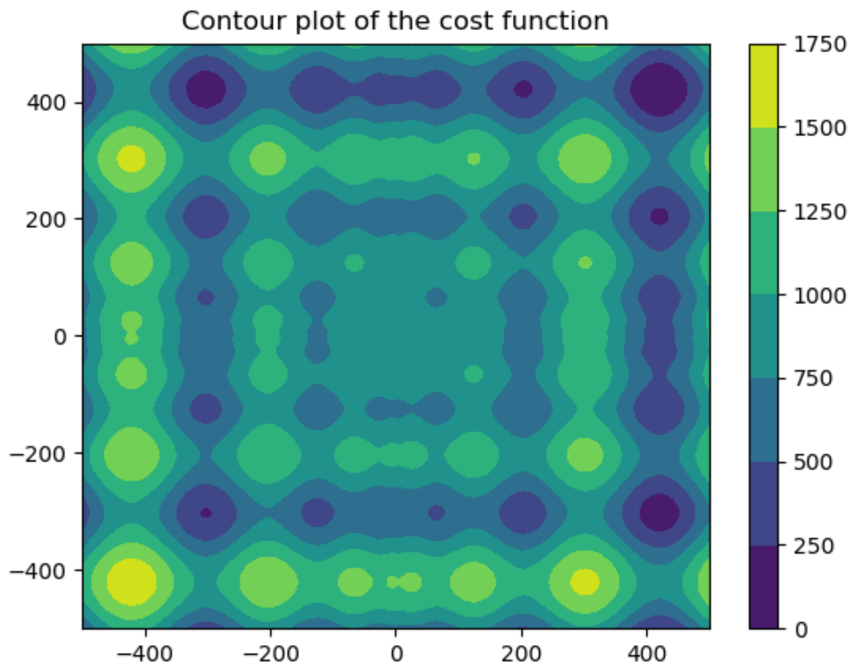
- Given Cost function to optimize:

$$f(\vec{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin\sqrt{|x_i|}$$
$$x_i \in [-500, 500]$$

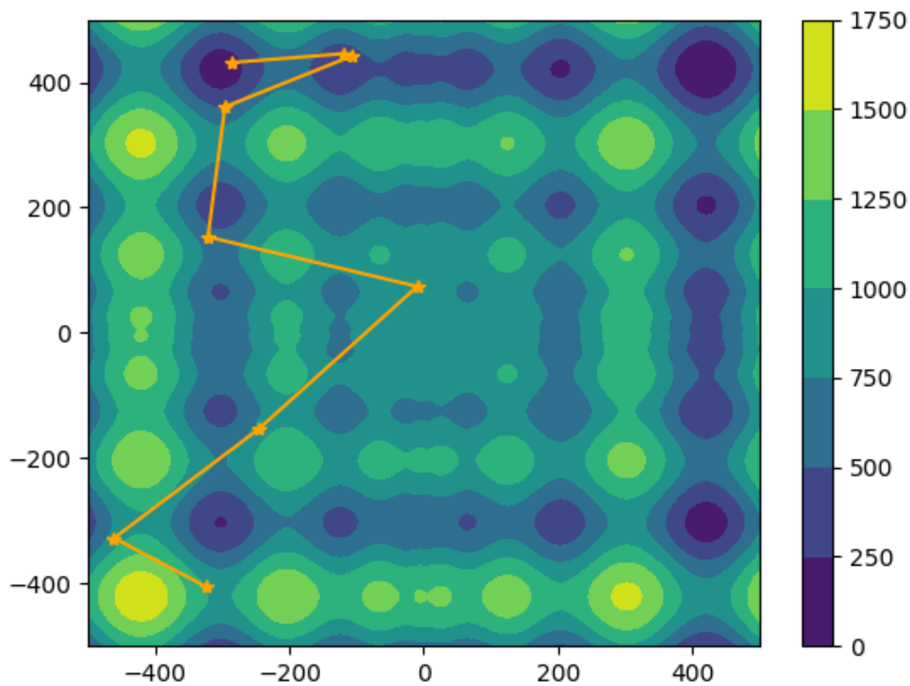
- Initialize maximum number of iterations and Cooling temperature
- Generate Initial samples from the given limits
- Choose a proposal pdf. It should be a symmetric pdf. I tried both Normal and Cauchy Distribution
- Generate the next sample given the previous sample from the proposal pdf
    - Meaning, the previous sample will be the mean if the proposal pdf is Gaussian
- Calculate the iteration temperature using a cooling method. The cooling method can be Logarithmic or Exponential or Polynomial. The iteration temperature depends on the initial temperature and the iteration number.
- Calculate the gibbs acceptance probability based on the new sample cost, previous sample cost and the iteration temperature
- If a random number generated is less than or equal to the acceptance probability, then accept the sample and append to the accepted list

Given Cost Function 3D plot and Contour plots: (Global Minimum lies in 420.9687, 420.9687)

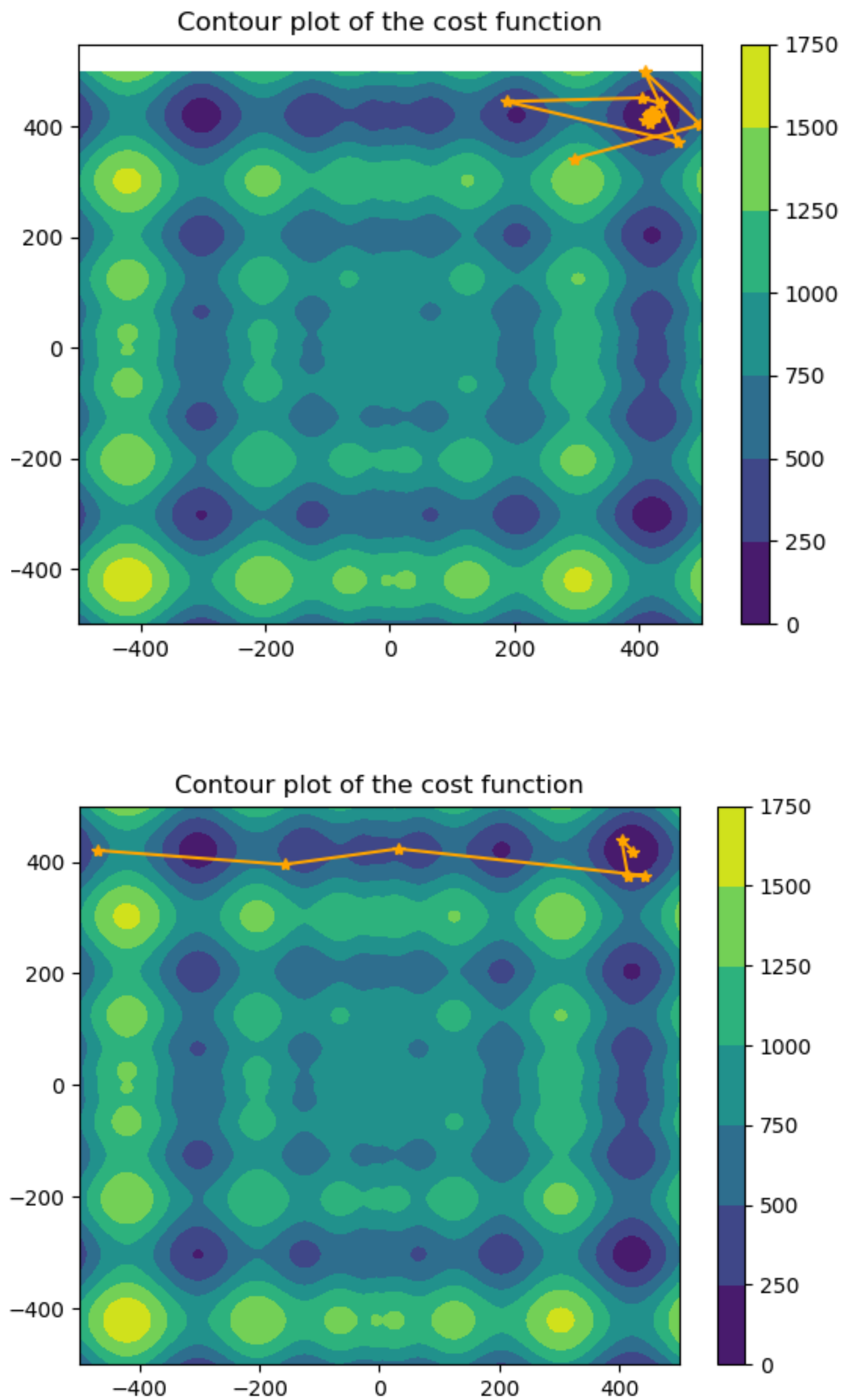




Some Experimental Results with Different Cooling Temperatures and Methods:



Sample plot where the global minimum convergence did not happen



Sample plots where the global minimum convergence happened

Discussion:

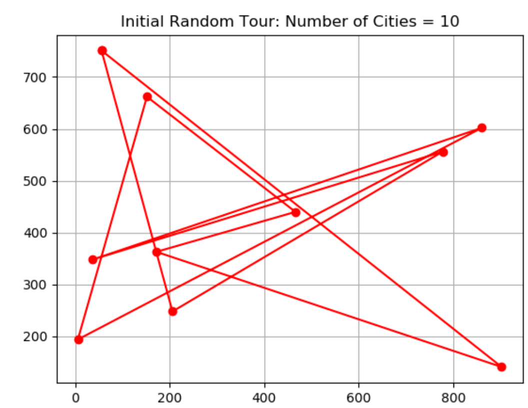
- The above experiments were conducted with various initial T values and various cooling functions
    - Polynomial
    - Exponential
    - Logarithmic
- Simulated Annealing is inspired from the Mechanical Annealing process where the material is heated to some temperature and then allowing it to cool
- Some of the best results are shown above. All the best results shown corresponds to:
    - Initial Temperature = 20
    - Maximum Iterations = 1000
    - Proposal Pdf = Normal with sigma = 200
    - Cooling Method = Exponential
- With these parameters, irrespective of the cooling method, the minimum converged to global minimum. Thus, with the best parameters like Initial Temperature and the right proposal distribution with the right parameters, Simulated Annealing converges to the Global Minimum
- As the Initial temperature is decreased, the converges was proved to be faster
- Simulated Annealing hence proved to be a method using which even though many local minimums were present in the cost function, right parameters allows the minimum to come out of the local minimum circle and converges to the global minimum

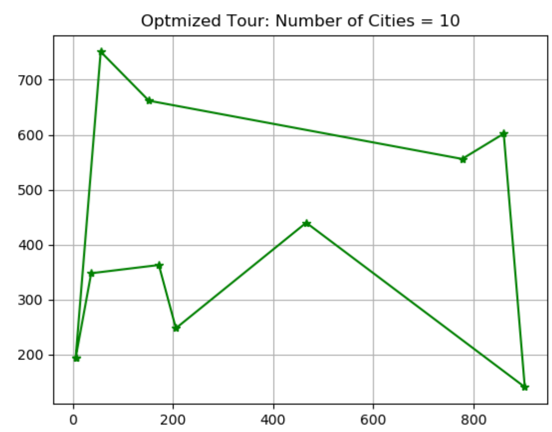
QUESTION 3 – FINDING OPTIMAL PATH

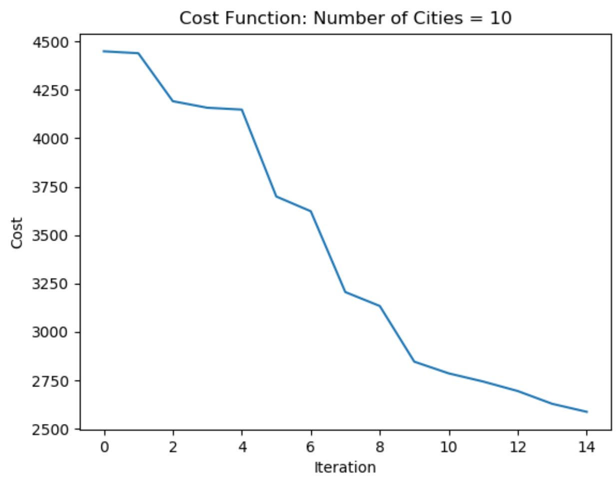
Markov Chain Monte Carlo Simulation for Finding Optimal Path:

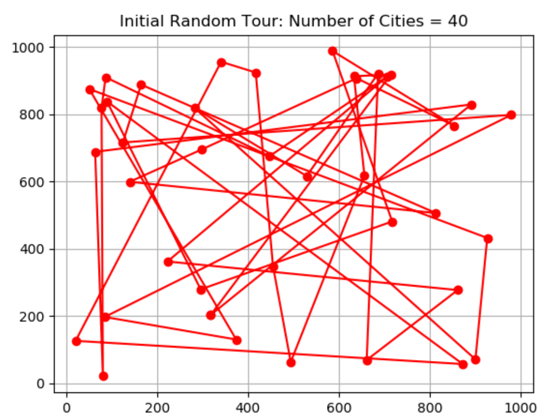
- Traveling Salesmen Problem as stated in the question is an NP-Hard Problem
- So, given a list of cities and the distances between each pair of cities, what is the shortest possible route for the Salesmen to visit each city once
- This problem can be solved by MCMC method using Simulated Annealing. The steps are elaborated below:
    - First, Get the cities location values based on the boundary conditions. Also initialize the Initial temperature
    - Generate a random tour itinerary
    - Select on a random two cities in from the cities list
    - Swap and get a swapped tour itinerary
    - Find the iteration temperature based on some Cooling Function
    - Find the cost of travel for both the Previous tour itinerary and Swapped tour itinerary
    - Using the cost of travel and iteration temperature, find the acceptance probability
    - If a random number generated is less than or equal to the acceptance probability, then the swapped tour is considered for the next iteration
    - Continue until the process converges or that there is no change in the tour itinerary
- The number of cities were changed from 10, 40, 400 and 1000

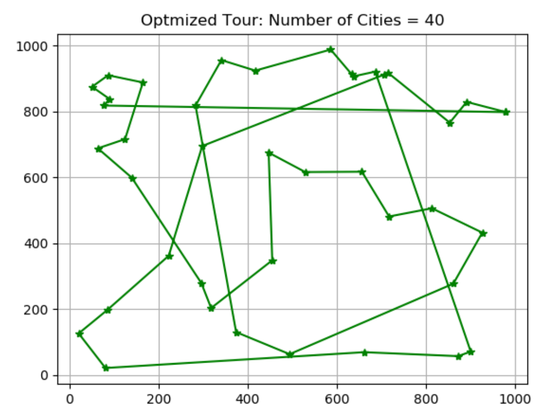
Results for Different Number of Cities:

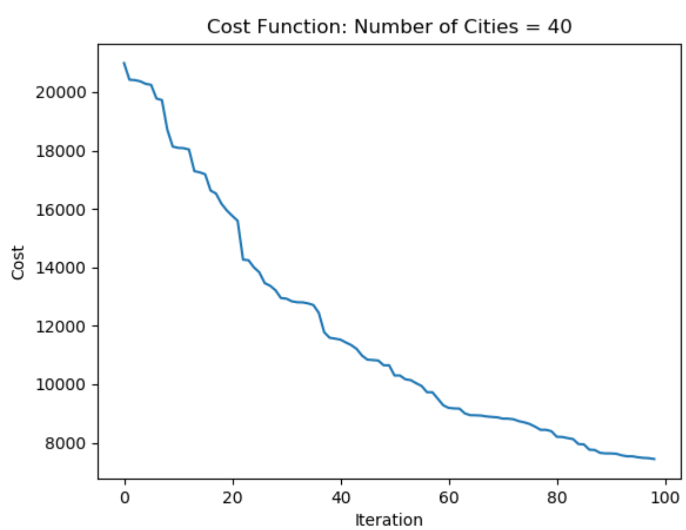


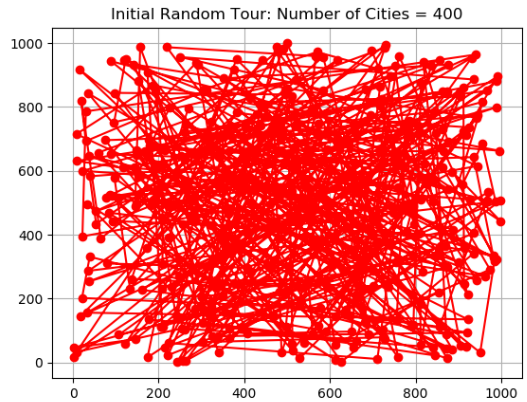


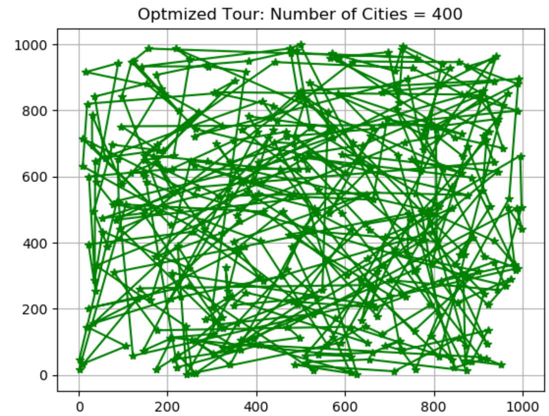

Cost Function: Number of Cities = 10
Cost
Iteration

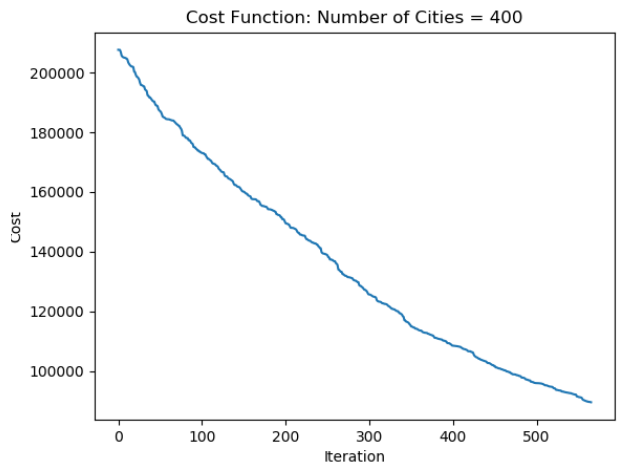

Initial Random Tour: Number of Cities = 40

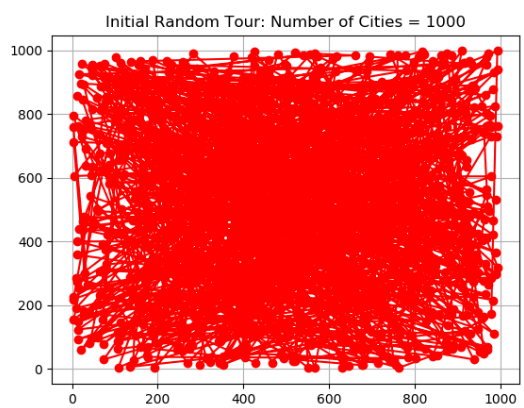

Optmized Tour: Number of Cities = 40

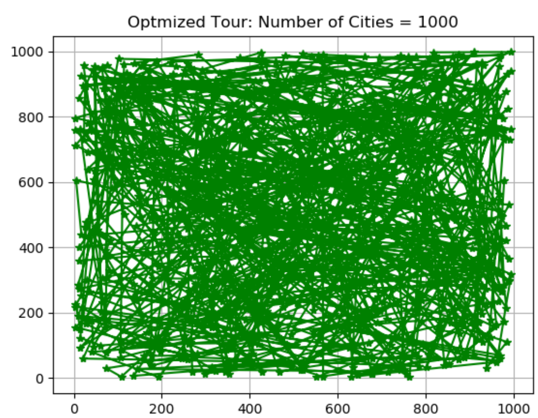

Cost Function: Number of Cities = 40
Cost
Iteration

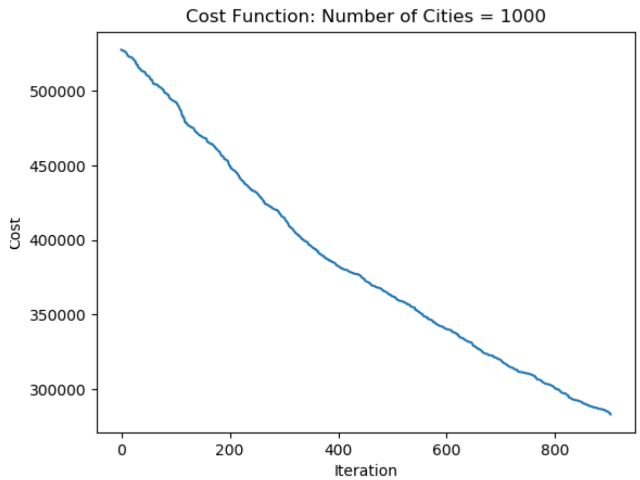

Initial Random Tour: Number of Cities = 400


Optmized Tour: Number of Cities = 400


Cost Function: Number of Cities = 400
Cost
Iteration


Initial Random Tour: Number of Cities = 1000


Optmized Tour: Number of Cities = 1000



DISCUSSION:

- The above plots show the Simulated Annealing method for finding optimal path in the traveling Salesman Problem
- As we can see when the number of cities is 10, the algorithm converged very easily. The optimal path can be seen very clearly converged above
- But as the number of cities increases, the convergence was not achieved.
- I tried changing the initial Temperature and cooling method, still the convergence did not happen for increasing values of number of cities.
- The cost function shows that there is a very slow and gradual decrease in the cost of travel. This trend is there even when the convergence did not happen. So, my guess is if the number of iterations is increased, even for larger number of cities, the Simulated Annealing method will converge. Just the time taken to converge will be very high.

CODES

QUESTION-1

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 19 22:24:07 2018

@author: abinaya
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta
from scipy.stats import norm
import scipy
#plt.close('all')

class Metropolis_Hastings():

    def __init__(self, proposal_pdf, sigma):
        self.proposal_pdf = proposal_pdf
        self.sigma = sigma

    def return_stationary_pdf(self,x):
        return ((0.6*beta.pdf(x,1,8)) + (0.4*beta.pdf(x,9,1)))

    def return_proposal_pdf(self,x,mu):
        if self.proposal_pdf == "Normal":
            return norm.pdf(x, loc=mu, scale=self.sigma)
        if self.proposal_pdf == "Cauchy":
            return scipy.stats.cauchy.pdf(x, loc=mu, scale=self.sigma)


    def generate_initial_sample(self):
       while True:
          sample = np.random.uniform(-1,1)
          sample_pdf = self.return_stationary_pdf(sample)
          if sample_pdf!= 0:
             print "Initial Chosen Sample: ",sample
             return sample
             break
```

```
    def proposal_step(self,previous_sample):
        if self.proposal_pdf == "Normal":
            return np.random.normal(loc=previous_sample, scale=self.sigma)
        elif self.proposal_pdf == "Cauchy":
            return scipy.stats.cauchy.rvs(loc=previous_sample)


    def acceptance_step(self, new_sample, previous_sample):
        num1 = self.return_stationary_pdf(new_sample)
        num2 = self.return_proposal_pdf(previous_sample,new_sample)
        den1 = self.return_stationary_pdf(previous_sample)
        den2 = self.return_proposal_pdf(new_sample,previous_sample)
        acceptance_probability = min(1, ((num1*num2)/(den1*den2)))
        return acceptance_probability

    def generate_samples(self, no_samples):
        samples_generated = []
        samples_generated_pdf = []
        initial_sample = self.generate_initial_sample()
        samples_generated.append(initial_sample)
        samples_generated_pdf.append(self.return_stationary_pdf(initial_sample))
        i=1
        while (i < no_samples):
            previous_sample = samples_generated[i-1]
            new_sample = self.proposal_step(previous_sample)
            acceptance_probability = self.acceptance_step(new_sample, previous_sample)
            if(np.random.uniform() <= acceptance_probability):
            #if(acceptance_probability > 0.5):
                samples_generated.append(new_sample)
                samples_generated_pdf.append(self.return_stationary_pdf(new_sample))
                i+= 1
        return samples_generated, samples_generated_pdf


trial = Metropolis_Hastings(proposal_pdf="Normal",sigma=10)
orig_pdf = []
x_samples = []
for x in np.arange(0,1, 0.01):
    x_samples.append(x)
    orig_pdf.append(trial.return_stationary_pdf(x))
plt.figure()
plt.plot(x_samples, orig_pdf)
plt.title('Given Stationary Mixed Distribution')
plt.xlabel('x')
```

```
plt.ylabel('pdf')

mh_samples_0_001 = Metropolis_Hastings(proposal_pdf="Cauchy",sigma=0.001)
samples_generated, samples_generated_pdf = mh_samples_0_001.generate_samples(5000)
plt.figure()
plt.plot(samples_generated,marker='*')
plt.title('Samples Generated - Proposal pdf = Cauchy; Sigma = 0.001')
plt.xlabel('Time')
plt.ylabel('Samples')

plt.figure()
plt.hist(samples_generated, bins=100)
plt.title('Histogram of samples - Proposal pdf = Cauchy; Sigma = 0.001')

###
mh_samples_0_01 = Metropolis_Hastings(proposal_pdf="Cauchy",sigma=0.01)
samples_generated, samples_generated_pdf = mh_samples_0_01.generate_samples(5000)
plt.figure()
plt.plot(samples_generated,marker='*')
plt.title('Samples Generated - Proposal pdf = Cauchy; Sigma = 0.01')
plt.xlabel('Time')
plt.ylabel('Samples')

plt.figure()
plt.hist(samples_generated, bins=100)
plt.title('Histogram of samples - Proposal pdf = Cauchy; Sigma = 0.01')

###
mh_samples_0_1 = Metropolis_Hastings(proposal_pdf="Cauchy",sigma=0.1)
samples_generated, samples_generated_pdf = mh_samples_0_1.generate_samples(5000)
plt.figure()
plt.plot(samples_generated,marker='*')
plt.title('Samples Generated - Proposal pdf = Cauchy; Sigma = 0.1')
plt.xlabel('Time')
plt.ylabel('Samples')

plt.figure()
plt.hist(samples_generated, bins=100)
plt.title('Histogram of samples - Proposal pdf = Cauchy; Sigma = 0.1')

###
mh_samples_1 = Metropolis_Hastings(proposal_pdf="Cauchy",sigma=1)
samples_generated, samples_generated_pdf = mh_samples_1.generate_samples(5000)
plt.figure()
```

```
plt.plot(samples_generated,marker='*')
plt.title('Samples Generated - Proposal pdf = Cauchy; Sigma = 1')
plt.xlabel('Time')
plt.ylabel('Samples')

plt.figure()
plt.hist(samples_generated, bins=100)
plt.title('Histogram of samples - Proposal pdf = Cauchy; Sigma = 1')

###
mh_samples_5 = Metropolis_Hastings(proposal_pdf="Cauchy",sigma=5)
samples_generated, samples_generated_pdf = mh_samples_5.generate_samples(5000)
plt.figure()
plt.plot(samples_generated,marker='*')
plt.title('Samples Generated - Proposal pdf = Cauchy; Sigma = 5')
plt.xlabel('Time')
plt.ylabel('Samples')

plt.figure()
plt.hist(samples_generated, bins=100)
plt.title('Histogram of samples - Proposal pdf = Cauchy; Sigma = 5')

###
mh_samples_10 = Metropolis_Hastings(proposal_pdf="Cauchy",sigma=10)
samples_generated, samples_generated_pdf = mh_samples_10.generate_samples(5000)
plt.figure()
plt.plot(samples_generated,marker='*')
plt.title('Samples Generated - Proposal pdf = Cauchy; Sigma = 10')
plt.xlabel('Time')
plt.ylabel('Samples')

plt.figure()
plt.hist(samples_generated, bins=100)
```

QUESTION-2

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 20 17:05:26 2018

@author: abinaya
"""
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

class Simulated_Annealing_Optimization():

    def __init__(self, proposal_pdf, sigma, T, max_iter, cooling_function):
        self.proposal_pdf = proposal_pdf
        self.sigma = sigma
        self.T = T
        self.max_iter = max_iter
        self.cooling_function = cooling_function

    def visualize_cost_function(self):
        n = 2
        x1 = np.arange(-500, 500, 0.1)
        x2 = np.arange(-500, 500, 0.1)
        x1mesh, x2mesh = np.meshgrid(x1, x2, sparse=True)
        z = (418.9829 * n) - ((x1mesh * np.sin(np.sqrt(abs(x1mesh)))) + (x2mesh *
np.sin(np.sqrt(abs(x2mesh)))))

        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot_surface(x1, x2, z, cmap=cm.coolwarm,linewidth=0, antialiased=False)
        plt.title('3D plot of the cost function')
        #plt.colorbar()

        plt.figure()
        plt.contourf(x1,x2,z)
        plt.title('Contour plot of the cost function')
        plt.colorbar()

    def return_cost_value(self,x1,x2):
        n = 2
        if ((x1 >= -500) & (x1 <= 500) & (x2 >= -500) & (x2 <= 500)):
            return (418.9829 * n) - ((x1 * np.sin(np.sqrt(abs(x1)))) + (x2 * np.sin(np.sqrt(abs(x2)))))

    def proposal_step(self,previous_sample):
        if self.proposal_pdf == "Normal":
            sample = np.random.normal(loc=previous_sample, scale=self.sigma)
            if sample[0] < -500:
                sample[0] = 500
            if sample[0] > 500:
```

```
            sample[0] = 500
        if sample[1] < -500:
            sample[1] = -500
        if sample[1] > 500:
            sample[1] = 500
        return sample
    if self.proposal_pdf == "Cauchy":
        return np.random.standard_cauchy(size=2)


def generate_initial_sample(self):
    initial_sample = np.random.uniform(-500,500, size=2)
    print "Initial Chosen Sample: ",initial_sample
    return initial_sample

def return_gibbs_acceptance_probability(self, new_sample, previous_sample, iterT):
    cost_previous_sample = self.return_cost_value(previous_sample[0], previous_sample[1])
    cost_new_sample = self.return_cost_value(new_sample[0], new_sample[1])
    #print "---"
    #print new_sample, cost_new_sample
    #print previous_sample, cost_previous_sample
    #print iterT
    acceptance_probability = min(1, np.exp(-1 * (cost_new_sample - cost_previous_sample)/
iterT))
    return cost_previous_sample, cost_new_sample, acceptance_probability

def return_iterTemp(self,n):
    if self.cooling_function == "Polynomial":
        return self.T * pow(n+1, -0.751)
    elif self.cooling_function == "Logarithmic":
        return self.T / np.log(n+1)
    elif self.cooling_function == "Exponential":
        return self.T / np.exp(n+1)


def optimize(self):
    print "Plotting given cost function --"
    #self.visualize_cost_function()
    samples_generated = []
    initial_sample = self.generate_initial_sample()
    samples_generated.append(initial_sample)
    i=1
    n=0
    while (n < self.max_iter):
```

```
            #print "--------------"
            previous_sample = samples_generated[i-1]
            #print previous_sample
            new_sample = self.proposal_step(previous_sample)
            #print new_sample
            iterT = self.return_iterTemp(i)
            #print iterT
            cost_previous_sample, cost_new_sample, acceptance_probability =
self.return_gibbs_acceptance_probability(new_sample, previous_sample, iterT)
            #print acceptance_probability
            if((cost_new_sample <= cost_previous_sample) or (np.random.uniform() <=
acceptance_probability)):
            #if(acceptance_probability > 0.5):
                #print True
                samples_generated.append(new_sample)
                i+= 1
                #n-= 1
            n+= 1
        return samples_generated



optimize_1 = Simulated_Annealing_Optimization("Normal", sigma=200, T=20, max_iter=1000,
cooling_function="Exponential")
samples_generated = optimize_1.optimize()
optimize_1.visualize_cost_function()

print "Converged to: ", samples_generated[-1]
samples_generated = np.array(samples_generated)
samples_generated = np.reshape(samples_generated, [len(samples_generated),2])
plt.plot(samples_generated[:,0], samples_generated[:,1], marker='*', color='orange')
```

QUESTION-3

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri May  4 19:07:34 2018

@author: abinaya
"""

import numpy as np
```

```
import matplotlib.pyplot as plt
import random

class Traveling_Salesmen_problem():

    def __init__(self, no_cities, T, cooling_function, max_iter):
        self.no_cities = no_cities
        self.T = T
        self.cooling_function = cooling_function
        self.max_iter = max_iter

    def generate_cities(self):
        return [random.sample(range(0,1000), 2) for x in range(self.no_cities)]

    def generate_initial_random_tour(self):
        return random.sample(range(self.no_cities),self.no_cities)

    def travel_cost(self, tour):
        cost = 0
        for i in range(0, self.no_cities-1):
            from_city_index = tour[i]
            to_city_index = tour[i+1]
            from_city = self.cities[from_city_index]
            to_city = self.cities[to_city_index]
            cost += np.linalg.norm(np.array(from_city) - np.array(to_city))
        return cost

    def return_iterTemp(self,n):
        if self.cooling_function == "Polynomial":
            return self.T * pow(n+1, -0.751)
        elif self.cooling_function == "Logarithmic":
            return self.T / np.log(n+1)
        elif self.cooling_function == "Exponential":
            return self.T / np.exp(n+1)
        elif self.cooling_function == "Other":
            return self.T * 0.99

    def generate_acceptance_probability(self, tour_cost, swapped_tour_cost, iterT):
        return np.exp((tour_cost - swapped_tour_cost) / iterT)


    def solve_problem(self):
        self.cities = self.generate_cities()
        tour = self.generate_initial_random_tour()
```

```
    initial_tour = tour[:]

    cost_to_plot = []

    for i in range(0,self.max_iter):
      swap_positions =  random.sample(range(self.no_cities),2)
      swapped_tour = tour[:]
      swapped_tour[swap_positions[0]] = tour[swap_positions[1]]
      swapped_tour[swap_positions[1]] = tour[swap_positions[0]]

      tour_cost = self.travel_cost(tour)
      swapped_tour_cost = self.travel_cost(swapped_tour)

      iterT = self.return_iterTemp(i)

      acceptance_probability = self.generate_acceptance_probability(tour_cost,
swapped_tour_cost, iterT)
      if(np.random.uniform() <= acceptance_probability):
        tour = swapped_tour[:]
        cost_to_plot.append(swapped_tour_cost)

    return initial_tour,tour, cost_to_plot



no_cities = 400
tr_sa_pr = Traveling_Salesmen_problem(no_cities, 1, "Other", 5000)
initial_tour,tour, cost_to_plot = tr_sa_pr.solve_problem()
oldx = zip(*[tr_sa_pr.cities[initial_tour[i % no_cities]] for i in range(no_cities+1)])[0]
oldy = zip(*[tr_sa_pr.cities[initial_tour[i % no_cities]] for i in range(no_cities+1)])[1]
newx = zip(*[tr_sa_pr.cities[tour[i % no_cities]] for i in range(no_cities+1)])[0]
newy = zip(*[tr_sa_pr.cities[tour[i % no_cities]] for i in range(no_cities+1)])[1]

plt.figure()
plt.plot(oldx, oldy, marker='o', color='r')
plt.grid('on')
plt.title('Initial Random Tour: Number of Cities = '+str(no_cities))

plt.figure()
plt.plot(newx, newy, color='g', marker='*')
plt.grid('on')
plt.title('Optmized Tour: Number of Cities = '+str(no_cities))

plt.figure()
```

```
plt.plot(cost_to_plot)
plt.xlabel('Iteration')
plt.ylabel('Cost')
plt.title('Number of Cities = '+str(no_cities))
```