CSCE 451 Lab2

Due July 15 at 3:00pm

A simple Simulated Annealing implementation in Python

In this lab we implement and investigate, the simulated annealing algorithm for numerical optimization.

Task 1: Understanding the Algorithm

We use the same algorithm described in the lecture. Following is the same algorithm with more details for the actual python implementation. Please review the code and make sure you understand it.

Simulated Annealing:

```
Let $s = s_0$
For $k=0$ through $k_{max}$ (exclusive):
    $T$ := temperature ($k/k_{max}$)
    Pick a random neighbour, $s_{new}$ := neighbour(s)
    If P(E(s),E(Snew),T) ≥ random(0,1):
        S := $s_{new}$
Output: the final state s
```

Task 2: Basic but generic Python code

Let us start with a very generic implementation. Please read the code and make sure you understand the main function. Once you feel comfortable with the code create a python file (or any other way you usually write your python code) and paste code there.

```
def annealing(random_start,
              cost_function,
              random_neighbour,
              acceptance,
              temperature,
              maxsteps=1000,
              debug=True):
    """ Optimize the black-box function 'cost_function' with the simulated annealing
algorithm."""
    state = random_start()
    cost = cost_function(state)
    states, costs = [state], [cost]
    for step in range(maxsteps):
        fraction = step / float(maxsteps)
        T = temperature(fraction)
        new_state = random_neighbour(state, fraction)
        new_cost = cost_function(new_state)
        if debug: print("Step #{:>2}/{:>2} : T = {:>4.3g}, state = {:>4.3g}, cost = {:>4.3g},
new_state = {:>4.3g}, new_cost = {:>4.3g} ...".format(step, maxsteps, T, state, cost, new_state,
new_cost))
        if acceptance_probability(cost, new_cost, T) > rn.random():
            state, cost = new_state, new_cost
            states.append(state)
            costs.append(cost)
            # print("  ==> Accept it!")
        # else:
        #    print("  ==> Reject it...")
    return state, cost_function(state), states, costs
```

Task 3: Other maintenance functions:

```
interval = (-10, 10)

def f(x):
    """ Function to minimize."""
    return x ** 2

def clip(x):
    """ Force x to be in the interval."""
    a, b = interval
    return max(min(x, b), a)

def random_start():
    """ Random point in the interval."""
    a, b = interval
    return a + (b - a) * rn.random_sample()

def cost_function(x):
    """ Cost of x = f(x)."""
    return f(x)
```

```
def random_neighbour(x, fraction=1):
    """Move a little bit x, from the left or the right."""
    amplitude = (max(interval) - min(interval)) * fraction / 10
    delta = (-amplitude/2.) + amplitude * rn.random_sample()
    return clip(x + delta)
```

```
def acceptance_probability(cost, new_cost, temperature):
    if new_cost < cost:
        # print("    - Acceptance probabilty = 1 as new_cost = {} < cost = 
{}...".format(new_cost, cost))
        return 1
    else:
        p = np.exp(- (new_cost - cost) / temperature)
        # print("    - Acceptance probabilty = {:.3g}...".format(p))
        return p
```

```
def temperature(fraction):
    """ Example of temperature dicreasing as the process goes on."""
    return max(0.01, min(1, 1 - fraction))
```

Task4: Run your code

Run your code and observe the result. Make sure your code works and convince yourself you understand the code and the output

```
annealing(random_start, cost_function, random_neighbour, 
acceptance_probability, temperature, maxsteps=30, debug=True);
```

Task5: Visualization

Comment previous code and add the code below. Here we save the output of annealing function in four variables that we use in see_annealing function for the visualization.
Note that the debug variable in turn to false here to prevent the annealing function priniting the debug information

```
state, c, states, costs = annealing(random_start, cost_function, random_neighbour, 
acceptance_probability, temperature, maxsteps=1000, debug=False)
```

Add functions below:

```
def see_annealing(states, costs):
    plt.figure()
    plt.suptitle("Evolution of states and costs of the simulated annealing")
    plt.subplot(121)
    plt.plot(states, 'r')
    plt.title("States")
    plt.subplot(122)
    plt.plot(costs, 'b')
    plt.title("Costs")
    plt.show()
```

```
def visualize_annealing(cost_function):
    state, c, states, costs = annealing(random_start, cost_function, random_neighbour, 
acceptance_probability, temperature, maxsteps=1000, debug=False)
    see_annealing(states, costs)
    return state, c
```

```
visualize_annealing(lambda x: x**2)
```

Task6:
Run the program and observe the output plots.
Run the code for each function below and paste the output (plot image) in a word doc. Submit the result with your code to BB.

1- $y = x^2$
2- $y = x^3$
3- $y = \cos(x)$
4- $y = \text{abs}(x)$