1 Background

Roughly speaking *simulated annealing* is an stochastic method for finding a reasonably good solution to an optimization problem. Depending on the problem this solution may be the best one, but that requires a previous knowledge of the system at hand.
In general we want to find a solution to a problem that minimizes a cost. For example, imagine we want to visit n cities $A_1, A_2, \ldots, A_n$. Clearly we would like to design an travel schedule that minimizes the overall cost of the trip. For small n this problem is easily solved, but when the numbers of cities increases we need a computational tool to find the best solution.
Simulated annealing is a surprisingly simple algorithm to solve this problem. It starts with a possible optimal solution and calculates the cost associated to such solution, let us call that number c. It then generates a new solution c', we have then two possibilities

- $c' < c$: the new solution is cheaper and prefer it over the previous one
- $c' > c$: the new solution is more expensive than the old one. In this case we randomly select between the two. More specifically, if

$$e^{(c-c')/T} > \mu \tag{1}$$

  we choose c', otherwise we keep c as the optimal solution. μ in this case is a random number between sampled from a uniform distribution in the domain 0 and 1.

This procedure is repeated many times while decreasing the temperature T.
In this session we will implement the annealing algorithm to minimize the cost function

$$c(x) = e^{-(x-1)^2} \sin(8x) + 1 \tag{2}$$

2 Problems

(P1) Create a function called `cost` that returns the cost for a given solution `solution`. In this case, the cost is given by Eq. (2)

```
cost(solution)
```

(P2) Make a plot of the cost for solutions in the range -3 o 3, and identify a range for the optimal value.

(P3) Create a function that returns the a new solution given a possible one.

```
def neighbor (solution):
    step_size = 1.0
    return (2 * np.random.random() - 1) * step_size + solution
```

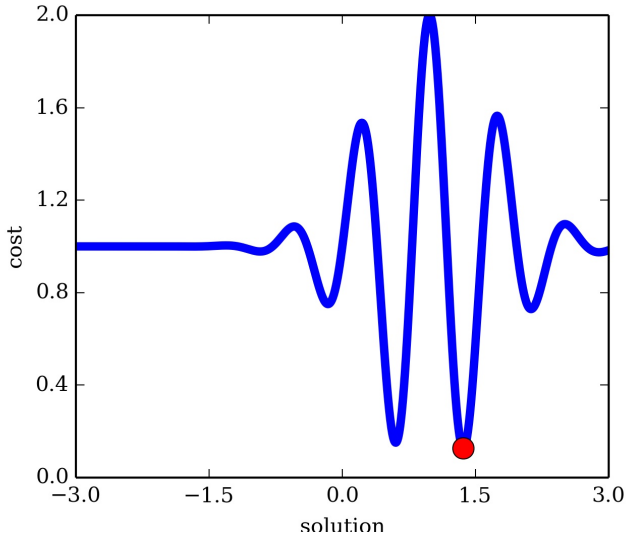
(P4) Start with a randomly selected solution in the range found in step (P2). Fix $T = 1$ and $T_{\text{min}} = 10^{-5}$

(P5) Generate a new solution with the function `neighbor`.

(P6) Decide which of the two solutions is the best one based on what we discussed earlier (See Eq. 1)

(P7) Repeat steps (P5) & (P6) 100 times

(P8) Decrease T by a factor of 0.9, and repeat steps (P5)-(P7) until the temperature drops below $T_{\min}$



3 Solutions

```
# define constants
T = 1.0
Tmin = 1e-5
fT = 0.9


# returns the cost for a given solution
def cost (solution):

    return np.exp(-((solution - 1.0)**2.0)) * np.sin(8 * solution) + 1

# returns the new position based on a given solution
def neighbor (solution):

    step_size = 1.0
    return (2 * np.random.random() - 1) * step_size + solution


# initializing
s = np.random.uniform(0, 2)
c = cost(s)

while T > Tmin:
    for i in range(100):

        s1 = neighbor(s)
        c1 = cost(s1)
        mu = np.random.random()

        if np.exp((c - c1) / T) > mu:
            s = s1
            c = c1

    T = fT * T

print 'Optimal solution: %f' % (s)
print 'Cost of optimal solution: %f' % (c)
```

```
>>> Optimal solution: 1.362231
>>> Cost of optimal solution: 0.127152
```