Homework 3
CS 169/268 Optimization
Fall 2018
Due: Thursday October 25 11:59pm on Canvas

As in HW1 and HW2, write your own code.

Global Optimization by Simulated Annealing

1. (UG & Grads) Implement as a black box optimizer (function evaluations only) the simulated annealing algorithm for binary-valued variables. See the code template at the end of this document. Your filled-in version of this template, along with any helper functions, must be in its own .py file, separate from any numerical tests you run.

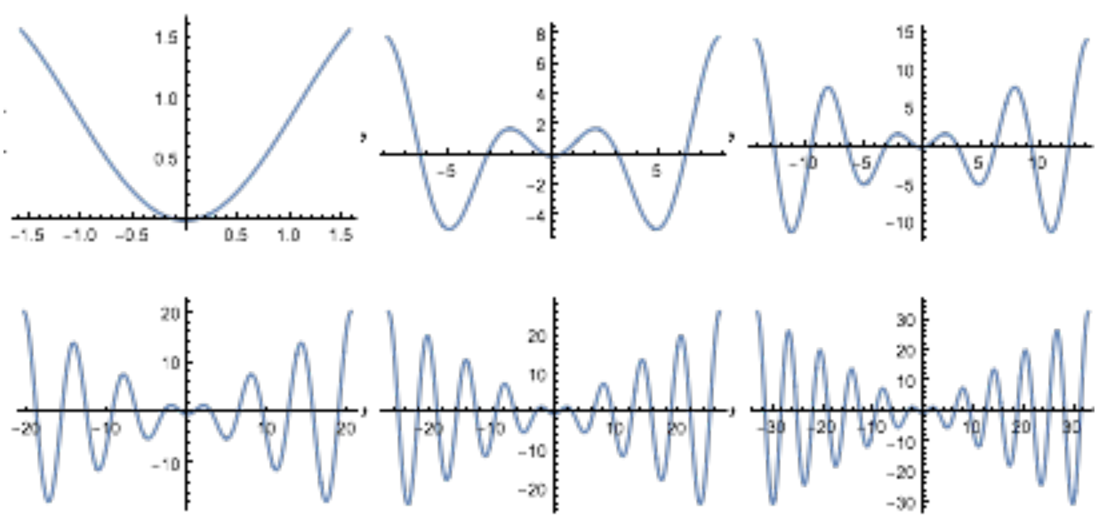
   (a) Test it quantitatively on a scalable combinatorial optimization problem (such as graph bisection, or 0/1 knapsack with a large penalty for overfilling the knapsack, or graph-2-coloring). Up to what problem sizes are you able to get it to produce reasonable answers?

   (b) Generalize the code to handle multiple discrete values. Test on such a problem, eg. graph partitioning or graph coloring.

2. (Grads and UG extra credit) Generalize your SA code to continuous real-valued variables. Compare it quantitatively on a problem of your choice to a local black-box optimization method such as coordinate descent (homework 1) or the Nelder-Mead method (which you would have to program up). If you want SA to win, try it on a “multimodal” optimization problem with many local minima such as:

$$f(x) = x\sin(x), \text{ in the interval } [-(2n+1/4)\pi, (2n+1/4)\pi], n \in \mathbb{N}$$

which for n=0,1,2,3,4,5 look like:



Extra credit: Choose the Nelder-Mead option in problem 2.

Code template for Simulated annealing:
As in HW1 and HW2, you may only modify this method inside the section marked out by dotted lines. You may define other helper functions, but your simulated_annealing method must be callable as in the __main__ method below. You may also add keyword arguments, but they must have default values.

```
def simulated_annealing(func, x0, allowed, t0, t_final):
    #simulated annealing routine.
    #inputs:
    #   func - the objective function
    #   x0 - the initial state
    #   allowed: a list, where allowed[i] is the list of allowed values for x[i].
    #       allowed[i] is the empty list if x[i] is a real-valued variable, rather than
    #       a discrete variable.
    #   t0: the initial temperature
    #   t_final: the final temperature
    #--------------------------------
    #STUDENT CODE GOES BELOW:
    curr_temp = t0
    curr_state = x0
    curr_score = func(curr_state)
    best = x0
    best_score = curr_score
    while curr_temp > ending_temp:
        curr_temp = curr_temp - 0.01
    #STUDENT CODE ENDS
    #--------------------------------
    return best, best_score

if __name__ == '__main__':
    items = [(10.0, 3.3),(1,.001),(1,.005),(1,5.0),(3.0,.01),(5.0,3.0)]
    def testf(x):
        max_weight = 10.0
        total_profit = -1.0*sum([x[i]*items[i][0] for i in range(len(items))])
        total_weight = sum([x[i]*items[i][1] for i in range(len(items))])
        if total_weight > max_weight or x[-1] < 0.05:
            return 1.0e12
        else:
            return total_profit/x[-1]
    allowed = [[0,1] for i in range(len(items))] + [[]]
    starting_temp = 10.0
    ending_temp = 0.1
    print(simulated_annealing(testf, [0,0,0,0,0,0,1.5], allowed, starting_temp, ending_temp))
```