Data Science @UL-FRI

Homework 4

Matej Kalc

1. Nelder-Mead

1.1 Implementation

I implemented the method using Python. My implementation can be found in nelder-mead.py.

1.2 Comparison of Nelder-Mead with other methods

Below we see the results of the Nelder-Mead method using different starting points and different diameters for starting point (for our case 1,2 and 3). My observations are that the method runs faster then some other methods like AdaGrad or Newton method. In all cases Nelder-Mead returned the minimum of each function. Even if it is extremely simple, it is really effective. Neldear-Mead performs better then GD, PGD and Nesterov (when comparing results from my HW2).

	Starting pos.	Diamet.	x*	f(x*)
f1	[0, 0, 0]	1	[-0.17 -0.23 0.17]	-0.2
	[1, 1, 0]		[-0.17 -0.23 0.17]	-0.2
	[0, 0, 0]	2	[-0.17 -0.23 0.17]	-0.2
	[1, 1, 0]		[-0.16 -0.23 0.17]	-0.2
	[0, 0, 0]	3	[-0.17 -0.23 0.17]	-0.2
	[1, 1, 0]		[-0.17 -0.23 0.17]	-0.2
f2	[1.2, 1.2, 1.2]	1	[1. 1. 1.]	0
	[-1, 1.2, 1.2]		[1. 1. 0.99]	0
	[1.2, 1.2, 1.2]	2	[1. 1. 1.]	0
	[-1, 1.2, 1.2]		[1. .99 0.99]	0
	[1.2, 1.2, 1.2]	3	[1. 1. 1.]	0
	[-1, 1.2, 1.2]		[1. 1. 0.99]	0
f3	[1, 1]	1	[2.99 0.5]	0
	[4.5, 4.5]		[3.01 0.5]	0
	[1, 1]	2	[3.02 0.51]	0
	[4.5, 4.5]		[2.99 0.5]	0
	[1, 1]	3	[3.01 0.5]	0
	[4.5, 4.5]		[3. 0.5]	0

Table 1. Results of the Neldear-Mead methodon on the functions from HW2. Values have been rounded to 2 decimal places.

2. Black box optimization

My student id is 63180368. Let f_1, f_2, f_3 be the black box function with i = 1, i = 2 and i = 3 respectively. Results of the Neldear-Mead method are shown in Table 2.

	x^*	$f(x^*)$
f_1	[0.87 0.27 0.78]	0.87
f_2	[1.16 0.11 0.73]	0.92
f_3	[0.15 1.06 1.08]	0.89

Table 2. Results of the Neldear-Mead methodon on the Black box functions. Values have been rounded to 2 decimal places.

I also tried the L_BFGS implementation from scipy. The results are shown in Table 3.

	x^*	$f(x^*)$
f_1	[0.86 0.31 0.81]	0.86
f_2	[0.81 0.31 0.86]	0.86
f_3	[0.31 0.81 0.86]	0.86

Table 3. Results of the L_BFGS on the Black box functions. Values have been rounded to 2 decimal places.

We observe that BFGS beat Nelder-Mead, but the difference are pretty small. We also observe that probably the black box functions are all the same but with indexes.

3. Local search study

3.1 Maximal and perfect matching

A matching M of graph G is said to maximal if no other edges of G can be added to M. A matching M of graph G is said to be a perfect match, if every vertex of graph G is incident to exactly one edge of the matching M, i.e., deg(V) = 1 for $\forall$ V. The degree of each and every vertex in the sub graph must have a degree of 1.

A naive approach to finding a maximal matching can be outlined as follows. Traverse the edges of a graph in depth-first order starting from an arbitrary node. Greedily match the first pair of nodes, and all possible node pairs during the traversal. On completing a traversal, record the matching and its edge count. Then, backtrack to the last branch to find a new matching. Continue until all matchings and their cardinals have been found.

If the number of vertexes id odd then the graph can not have a perfect matching, otherwise it can.

3.2 Fractional matching as a convex combination of matchings

Let M_i be a matching with only the i-th edge. We can construct a convex combination using $\alpha_i \in [0,1]$, where $\sum_{i \in E} \alpha_i = 1$.

3.3 Relax Maximal Weight Matching

I used the library CVXOPT. The used matrices for the solver are vector c, which contains the weights of all the edges, matrix G and vector h, which assures that $\sum_{e \in E(v)} x_e \leq 1$ and that $0 \leq x_e \leq 1$ for all edges. Since CVXOPT tries to minimize the cost we used negative weights, as shown in c. Example for n = 2 is shown below.

$$G = \begin{bmatrix} 1. & 0. & 1. & 0. \\ 1. & 0. & 0. & 1. \\ 0. & 1. & 1. & 0. \\ 0. & 1. & 0. & 1. \\ 1. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ -1. & 0. & 0. & 0. \\ 0. & -1. & 0. & 0. \\ 0. & 0. & -1. & 0. \\ 0. & 0. & 0. & -1. \end{bmatrix} \tag{1}$$

$$h = \begin{bmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} \quad c = \begin{bmatrix} -1.4359949 \\ -1.02592623 \\ -1.54966248 \\ -1.43532239 \end{bmatrix} \tag{2}$$

The maximal cost achieved with CVXOPT is 337.58. The optimal solution x^* contains only zeros and ones. The returned optimal solution is a proper matching.

3.4 Maximal Weight Matching

I implemented local optimization with k = 1,2 and 3 and performed 100000 steps.

k	cost
1	283.32
2	285.05
3	292.66

Table 4. Results of Local optimization with k = 1,2 or 3.

We observe that we are not that near the optimal cost value (see Table 4). We can improve our results by removing edges with a lower weight and adding edges with a higher weight. The probabilities for removing and adding the edges are calculated using equations 3 and 4.

$$p_e = \frac{2 - w_e}{\sum_{e \in E} 2 - w_e} \tag{3}$$

$$p_e = \frac{w_e - 1}{\sum_{e \in E} w_e - 1} \tag{4}$$

k	cost
1	311.73
2	301.22
3	303.75

Table 5. Results of Local optimization with optimized edge selection.

Including the weight information for each edge improved drastically our results (see Table 5). Local optimization with k = 1 gave the best result with a cost value of 311.73. The difference between the optimal cost value and the cost value of k = 1 is 25.85.