This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth one-tenth the number of points of a regular assignment.

PIC 16, Winter 2018 – (Optional) Assignment 8Wb

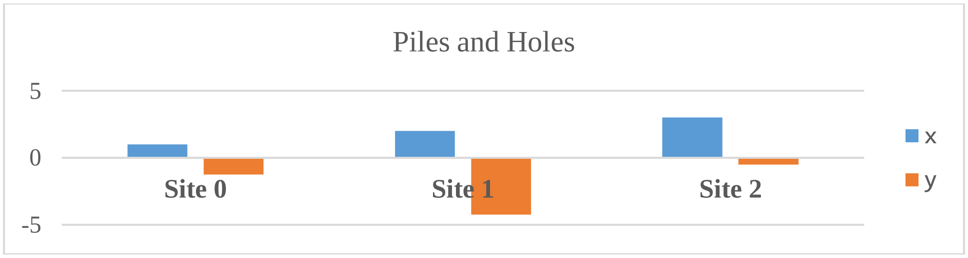
Assigned 2/28/2018. Code (a single `.py` file) due by the end of class 3/5/2018 on CCLE. Hand in a printout of this document with the self-assessment portion completed by the end of class on 3/5/2018.

In this assignment, you will calculate the Earthmover's Distance by expressing it as a linear programming problem.

As the primary purpose of this problem is to practice converting a word problem into a linear programming problem, I suggest that you do **not** *research Earthmover's Distance before trying to formulate the problem yourself. While there are many equivalent problem formulations that will result in the same value of the Earthmover's Distance, you will only receive full credit if your problem formulation aligns directly with the problem statement presented herein.*

Task

Consider two 1D arrays, `x` and `y`. Assume that the arrays are both of size N, `x` contains only non-negative values, `y` contains only non-positive values, and the sum of all the elements in `x` and `y` *combined* is zero. We can visualize each element of `x` as a pile with the specified amount of dirt ("earth"), and each element of `y` as a hole that can hold the specified amount of dirt. For example, if `x = [1, 2, 3]` and y = `[-1.5, -4.25, -0.25]`, we can visualize piles as blue bars and holes as orange bars like:



Our task is to move the dirt around from the piles `x` to fill all the holes `y`. We want to know how much "work" it takes to do so. If all the "job sites" (each with a pile and a hole) are collocated, then there isn't much strategy to it. But if the sites are separated, and if the work is calculated as the dirt amount multiplied by the distance traveled, we have some choices to make: how much dirt from each pile should we move into each hole to minimize the total amount of work? This minimum amount of work is called the Earthmover's Distance between the arrays. When $N = 3$ as above, the problem may be easy to solve, but the optimal strategy may not be so obvious when there are many job sites.

We'll consider the special case in which the job sites are arranged in a line and evenly spaced with distance 1 unit; e.g. to move a unit of dirt from the pile at site 1 to the hole at site 1 takes 0 work, from pile 1 to hole 2 takes 1 unit of work, and from pile 2 to hole 0 takes 2 units of work.

Keeping in mind the following:

- The amount of dirt moved *from* each pile of `x` *cannot exceed* the amount of dirt in the pile.
- The amount of dirt moved *to* each hole of `y` must *exactly* fill the hole.
- The work done is the product of dirt moved and the distance it traveled.

Write a function that accepts inputs arrays `x` and `y` calculates the Earthmover's distance between them.

This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth one-tenth the number of points of a regular assignment.

Additional hints and specifications:

Use `scipy.optimize.linprog` inside your function to calculate the minimum work.

We haven't covered `linprog` specifically; you're expected to use the documentation to figure out the syntax. If you understand what we've done with `minimize`, you already have the concepts you need for `linprog`.

Required (for 10% credit): compare the performance of the two available methods (see documentation). Which is faster for large problems (*n*~100)? Do they give the same answer, or does one fail? *Note: I wrote the interior point method and contributed it to Scipy because I found that the existing simplex implementation was unreliable. (In theory, the simplex* algorithm *can solve any linear programming problem; the implementation in Scipy is buggy.) In testing, I have found the interior-point method to be faster for all but the smallest, simplest problems. (However, it sounds like there are better implementations of the simplex algorithm that are competitive with the best interior point implementations, at least for certain types of problems.)* Summarize your findings.

Required (for 10% credit): approximately what percent of the elements in your constraint matrices are zero and how does it depend on the problem size? Answer below:

Required (for 10% credit): for the interior point method, try the sparse option. *Note: a sparse problem has a high percentage of zeros. Sparse matrix techniques take advantage of the sparsity. They save space by not storing the zeros in the matrix – they record only the nonzero elements and where they are located. Performing operations between matrices stored like this, they save time by avoiding explicit operations involving zeros.* How much does it speed up the solution of large problems?

You can check your function with PyEMD, a module that you'll have to install in order to use. PyEMD is distributed with Anaconda but not installed by default, so you can use `conda install pyemd` to get it. See the documentation for reference. *Note: the "histograms" are just the arrays* `x` *and* `y` *and the distance metric is just the objective function coefficients (*`c`*) reshaped as a square array. Also, expect PyEMD to be much faster than* `linprog`*, as it uses an algorithm specialized for this particular type of problem.*

To test your function (and to practice using PyEMD), start with very simple cases for which you can intuit the answer. Once you understand how PyEMD is working, you can check the results of your function against it. I suggest testing with large (e.g. `n = 100`), randomly generated arrays before submitting. Remember that without normalization, the sums of randomly generated arrays will not be equal (unless you are absurdly lucky).

For the remaining credit: do the results of your method always agree with the results of PyEMD? ______

(Agreement does not need to be exact. Mine typically agrees within ~ 0.01%)