ECI 289I: Applied Evolutionary Computing
Assignment #6

(1) Statistical comparison of MOEAs

(a) Perform the following experiment:

- Choose 3 algorithms to compare from the Platypus library. If you don't have a preference, try NSGAIII, eMOEA, and GDE3 with their default settings.
- Optimize the 2-objective DTLZ2 problem
- Run each algorithm for 30 random seeds, 1000 function evaluations each
- Calculate the hypervolume for each random seed

This will take some time to run, so you should save results in a file (look up `np.savetxt`) so that you can work on the rest of this problem without re-running the experiment every time.

Create histograms of your hypervolume results for each algorithm. Ideally, put all three histograms on the same axes. Do they look Gaussian? Which algorithm seems to do best?

(Matplotlib has a `plt.hist(x)` function, or if you have Seaborn installed, it can make some nice histograms with `sns.distplot(x)`, see here:
http://seaborn.pydata.org/examples/distplot_options.html)

(b) Perform a Mann-Whitney U test on all pairs of algorithms (3 in total). If any pairs are significantly different, perform a one-tailed test to see which is better. Use a critical value of $\alpha = 0.05$ across all pairwise experiments (i.e. use the Bonferroni correction to adjust α). Describe your findings. Are the results of these tests generalizable to other problems? Other NFE?

(2) Empirical complexity

In this problem we'll estimate the computational complexity of solving the TSP using our list-reversal code from a previous assignment. The mutation operation is shown on the next page, in case you need to update your version. (You can remove the simulated annealing part of the code).

(a) Write a function `get_random_cities(N)` that returns an instance of the TSP—a set of N cities randomly located on the square $[0,1]^2$.

(b) We want to calculate NFE-to-convergence as city size increases, and then estimate the order of this function (polynomial, exponential, etc.). To do this, we first need to estimate the optimal tour length (more reading here):
https://en.wikipedia.org/wiki/Travelling_salesman_problem#TSP_path_length_for_random_sets_of_points_in_a_square

Use the upper bound of $L^* = \sqrt{2N}$ as your measure of convergence.

For problem sizes $N = [50, 100, 200, 400, 800]$, perform 10 random trials each and measure the NFE required to converge. Again, this will take some time to run—save your results to a file.

(c) Create a scatter plot of NFE-to-converge vs. problem size, like we did in class. Experiment with log-scales on the x and y axes. Can you find a relationship that looks linear, and estimate the slope? Report your estimate of the complexity using big-O notation. Recall that the best-known exact solution requires $O(n^2 2^n)$—can we do better, if we're willing to accept an inexact solution?

```
trial_tour = np.copy(tour)
a = np.random.randint(N)
b = np.random.randint(N)

# reverse the sub-tour between A and B
if a > b:
  a,b = b,a
trial_tour[a:b] = trial_tour[a:b][::-1]
```