Homework #1

ME 471/571

All of your work for this assignment should be turned in using Jupyter Notebooks. Please turn in a PDF of the results run on your laptop or desktop, or cluster, if indicated. Your should develop one notebook for each problem.

Undergraduate students may do the Graduate student problems for extra credit.

1. (**Python lists and arrays vs. NumPy arrays**) Write a test suite that illustrates the timing differences between using a Python list, a Python array, and a NumPy array. Test your timing results on the following vector operations

   (a) Vector addition : $w = v + v$

   (b) Vector multiplication $w_i = v_i * v_i$, (i.e. element-wise multiplication)

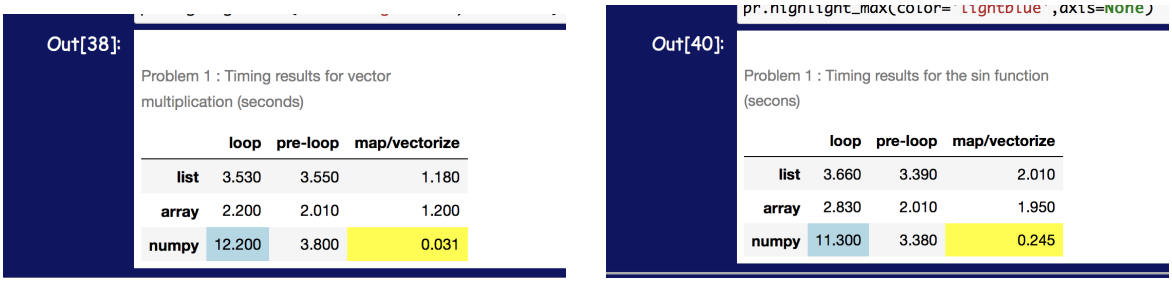
   (c) Elementary function $w = \sin(v)$. Use `math.sin(x[i])` for element-wise operations and `numpy.sin(x)` for array calculations.

   Your should produce the following results

   - For Python list and Python array objects, time the vector operations using a *loop* and the `map` function. For NumPy arrays, time the loop, and the vectorized operations, i.e. `w = v + v`.
   - Use the Python `%%timeit` magic command to get sufficiently accurate results. This command will run your code several times to get average statistics for timing results.
   - Using Pandas, create one or more tables comparing the results for each operation, for each object type. Highlight maximum and minimum times.
   - Using Matplotlib, create one or more bar charts illustrating the same information.
   - Include timing results from equivalent MATLAB operations in your results.
   - (Optional) Include results showing the benefits of pre-allocating memory for the list or array w, when using a loop. For NumPy arrays, use `x.resize()` to resize the array.

   Provide a summary of the results and draw conclusions from what you found. What results stand out as exceptionally bad? Or exceptionally good?

   Your descriptive text should also explain the differences between the storage formats of lists vs. arrays, and possible explanations for the performance differences. See Figure 1 for a sample table.



Out[38]:

Problem 1 : Timing results for vector multiplication (seconds)

	loop	pre-loop	map/vectorize
list	3.530	3.550	1.180
array	2.200	2.010	1.200
numpy	12.200	3.800	0.031

Out[40]:

Problem 1 : Timing results for the sin function (secons)

	loop	pre-loop	map/vectorize
list	3.660	3.390	2.010
array	2.830	2.010	1.950
numpy	11.300	3.380	0.245

Figure 1: Sample tables created using Pandas in a Jupyter notebook for Problem 1, with maximum and minimum values highlighted (MATLAB results are not included, but should be included in your table, for comparison.)

2. (**Computational statistics**) Write routines to compute the minimum and maximum, and standard deviation, of a large array of random numbers.

   - Use Pandas to produce a table showing the timing results and efficiency of your algorithms.
   - Use Pandas to produce a loglog plots showing *speed-up* and *efficiency*. For each plot, plot your actual results, and the theoretically expected results.

   For this problem, you will use the Python `multiprocessing` module, and the NumPy`random` module to generate uniformly distributed random numbers.

   Note 1: *Speed-up* S_p is defined as follows :

   $$S_p = \frac{T_1}{T_p} \tag{1}$$

   where T_1 is the time a job takes on 1 processor and T_p is the time the same job takes on p processors. Theoretically, one expects $S_p \sim p$, although perfect speed-up is rarely achieved, except in "embarrassingly parallel" cases. In a plot, speed-up is shown as T_p (y-axis) verses p (x-axis) on a loglog plot. To compare to the theoretical results, include in your plot a plot of p verses p.

   Note 2: *Efficiency* E_p is defined as

   $$E_p = \frac{S_p}{p} \tag{2}$$

   Theoretically, one expects $E_p \sim 1$, but values above 0.7 are considered good. Often, efficiency is expressed as a percentage, i.e. 72%. In a plot, efficiency is shown as E_p verses p, in a semilogx plot, i.e. the x-axis is scaled logarithmically, but the y-axis is scaled linearly.

   Note 3: See course GitHub site for examples of using Pandas to create tables and plots.

   Graduate students Write a parallel algorithm to compute the median of an array of numbers. Produce the same timing results above.

3. (**Matrix-vector multiply**) Write a parallel algorithm to compute the product of a square matrix and a vector. Compare timing results for two different approaches

   - Distribute rows of the matrix across processors so that each process computes the dot products involving rows of the matrix with given vector in serial.
   - Distribute the dot product computations so that the loop over rows is done in serial.

   Display speed-up and efficiency in both a table and plots.

4. (**Approximate** π) Use a Monte-Carlo method to approximate the value of π. The algorithm is as follows.

   - Generate a sequence of N, $N \gg 1$ random numbers in $[-1, 1]$.
   - Count the number of values K in the sequence which are in the circle of radius 1, centered at the origin.
   - The ratio of the number of values in the circle to N should approach the ratio of the area of the circle to the area of the enclosing box. Use this to approximate π.

   Run this on a single processor for a sequence of N values and report the value and timing results. Then, run this using N points on p processors, gathering the results and reporting the timing and your approximation. You should be able to show that by using more processors, you can get a better approximation for essentially the same total time.