Eric's Python Notes

Eric A. Applegate

ARTICLE HISTORY
Compiled November 12, 2019

1. Changes Eric made to Nathan's Python files

- in `allocate` function within `allocate.py`, updated input arguments to include the `WSFlag` parameter as in Matlab and added "equal" and "brute force ind" allocations (see next bullet points).
- in `allocate.py`, added the `equal_allocation` function for equal allocation
- in `allocate.py`, added the `bfind_allocation_smart` function for brute force independent allocation
- in `utils.py`, edited the `calc_phantom_rate` function (it was not working and needed some import statements)
- in `utils.py`, added the `calc_bf_rate` function based on my Matlab code and how Nathan coded `calc_phantom_rate` in Python
- added `python_timing.py`. This file contains the functions from the `Timing` folder of the Matlab implementation that read in the problems we used in the journal article timing tables, calculate times for each allocation, calculate rates for each problem, record the times and rates in `.pickle` files, and summarize the data to re-create the timing tables. The file is set up such that it can be called from the terminal (i.e. `python python_timing.py`) to start executing (and be left to run however long it takes).

2. Work that needs to be done

- Find all places where `allocate` is called and update to include `WSFlag` where appropriate
- Debug Brute Force allocations and rate computations. After running the $d = 3, r = 10$ timing table row in Table 1 (below), it seems as though there's something wrong with the Brute Force rate calculations. If we ignore the times (we expected them to be longer than Matlab), the phantom rates in Table 1 are approximately what we saw in the corresponding table in the journal article. However, the brute force rates are off across the row and the "MVN True" and "MVN Ind." allocation columns are also quite different from the journal article table. This suggests that something is wrong with the brute force rate calculations.
- After making the above changes and verifying that brute force is correct, move on to update/test/debug `solve` and `testsolve`. Updates may include changing

any calls to `allocate` (see first bullet), ensuring that previous alpha-hat vectors are not used in the sequential iterations (a change we made in Matlab around the time Nathan left the project), and ensuring that the output is saved in some fashion (probably `.pickle` files).

- Figure out how to incorporate all of this into PyMOSO via the command line. Or will this be a stand-alone package separate from PyMOSO?

Table 1. PYTHON WITHOUT WARM-STARTS For 10 MORS problems generated by the fixed Pareto method, each with $d \geq 3$ objectives and $r \geq 10$ systems, the rest of the table reports: the median number of Paretos and phantoms, sample quantiles of the wall-clock time T to solve for each allocation $\boldsymbol{\alpha}$ in minutes (m) and seconds (s); the median rate of decay of the $\mathbb{P}\{\text{MC}\}$ calculated by brute-force ($Z^{\text{bf}}_{0.5}(\boldsymbol{\alpha})\times 10^5$) or by the phantom approximation ($Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$).

d	r	p	Med. $\lvert\mathcal{P}^{\text{ph}}\rvert$	Metric	MVN True	MVN Phantom	MO-SCORE	MVN Ind.	iMO-SCORE	Equal
3	10	5	11	Median T	6m 31s	3.75s	0.71s	7m 52s	0.26s	0s
				75th %-ile T	10m 32s	5.62s	1.05s	15m 13s	0.38s	0s
				$Z^{\text{bf}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	545.830	950.225	924.567	524.921	720.966	426.498
				$Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	510.551	950.211	924.543	519.867	695.679	393.455
	500	10	21	Median T	–	–	–	–	–	–
				75th %-ile T	–	–	–	–	–	–
				$Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	–	–	–	–	–	–
				Median T	–	–	–	–	–	–
	10,000	10	21	75th %-ile T	–	–	–	–	–	–
				$Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	–	–	–	–	–	–
4	5,000	10	42	Median T	–	–	–	–	–	–
				75th %-ile T	–	–	–	–	–	–
				$Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	–	–	–	–	–	–
	10,000	10	44	Median T	–	–	–	–	–	–
				75th %-ile T	–	–	–	–	–	–
				$Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	–	–	–	–	–	–
5	10,000	10	90	Median T	–	–	–	–	–	–
				75th %-ile T	–	–	–	–	–	–
				$Z^{\text{ph}}_{0.5}(\boldsymbol{\alpha})\times 10^5$	–	–	–	–	–	–

Computed in MATLAB R2017a on a 3.5 Ghz Intel Core i7 processor with 16GB 2133 MHz LPDDR3 memory.