Background

The purpose of this project is to implement risk in the exponentiated gradient and online lazy updates algorithms to examine the impact of risk as a parameter on portfolio return. Both were already implemented in MATLAB by Arindam Banerjee, Puja Das, and Nicholas Johnson, and have been ported to Python to access additional libraries and functionality.

Progress

The exponentiated gradient algorithm has been successfully ported over, as shown by testing on the NYSE dataset. Using a learning rate (η) of .05, the multiplicative gain given by MATLAB is 26.678, and the return in Python is 26.845. The error between the two values is attributed to floating point rounding. The online lazy updates algorithm also has been successfully ported over, albeit a few bugs that are discussed below. For a log weight of 10^{-6} (η) and a L1 norm weight of 10^{-6} (α) with no transaction cost ($\gamma = 0$), the returns are 2677.8% and 2683.6%. Again, the error is attributed to floating point rounding.

In the Python code, there are a few points of interest. In the W-Update portion of sparse_ports_admm.py, the w_temp update doesn't work as the MATLAB code does. Instead, w_temp already projects the weights onto a simplex, and so the find_y function does nothing. In addition, floating point errors present issues with iterations. There is only accuracy to the nearest thousandth for non-multiplicative values and to the nearest fifth for multiplicative updates, thus iterations to minimize the cost function are different and not effective debugging tools. Finally, a small point: Python indexing begins at 0, while MATLAB indexing begins at 1, so when debugging, it was necessary to compare day A in Python with day A+1 in MATLAB.

Aside from the development, the classroom is moving along at the expected pace. All material is likely to be completed on time.

Setbacks

The online lazy updates algorithm has taken longer than expected to port from MATLAB to Python. Currently, the Python algorithm does not work for all initial parameters. Instead, it sometimes fails and begins recording a 'NaN' value for wealth. Once this occurs, the algorithm computation slows down and must be manually stopped.

This project began with the two algorithms written only in MATLAB, with no Python code to reference. To transfer the code, modules from the libraries pandas and numpy were used. First, as the program was built to be scaled, compute_PRM was built to compute price-relative matrices. It does so by a date range and stocks as inputs and pulling the price-relatives from Yahoo Finance. However, this function is not in use as the NYSE dataset given has been used instead. The exponentiated gradient algorithm was built with only minor debugging required. However online lazy updates, as a more complex algorithm, posed many issues. The entire code was first ported over and directly translated into Python, but many syntactical errors and silent errors were present. Issues such as indexing differences, matrix mismatch, dimensionality errors, and Python-specific required syntax changes meant the ported over code had to be heavily modified to make it usable.

However, after most of the debugging has been complete, errors persist. The code was first analyzed by examining errors in the first 10 days of the algorithms, which was effective at first, but no longer works. The first 10 days, under all initial parameters, now match the MATLAB code within a floating-point error range, and in many cases the algorithm will proceed to completion without any issues. This scenario presents a difficult position, as most setbacks are now silent and don't appear often. For example, a bug fixed recently manifests itself only after Day 379, with a log weight (η) value of 0.1 and an L1 norm weight (α) of 0.00001 (in Python). The wealth growth and portfolio weights matched MATLAB, with no indication of error, until the specific day, when the Python implementation showed a growth in wealth from 1.44 to 40 in a single day, and continued at that growth rate until overflow and a wealth showed a 'NaN' value. This bug was silent and time consuming to find and patch, and, like many recent issues, is caused by a subtle difference in how the languages were designed.

In addition to bug finding, time must also be devoted to improving the code structure. In one instance, the variable α was reused had to be refactored throughout the code as to avoid confusion. Also, the math in MATLAB is compact but not conducive to understanding what how the algorithm works. It was necessary to split up operations and introduce new intermediate variables to improve readability. The code also needs proper in-line documentation through commenting, as it is dense and sometimes difficult to interpret the writers' intentions.

Expectations

The project expectations must be revised to reflect the difficulties of creating a working online lazy updates algorithm in Python. The initial projection to complete two implementations of risk in exponentiated gradient and online lazy updates is unreasonable given the time constraints. Instead, a more feasible path would implement both methods of risk constraints (reject a portfolio with too high of a risk and modify the cost function to reflect volatility) on the exponentiated gradient algorithm or only portfolio rejection on both the exponentiated gradient and the online lazy updates algorithm. In either case, online lazy updates would be expected to be fully debugged by the end of the semester. Another alternate path would be to only implement one risk constraint, but perform extensive testing and display results on various time periods with various stocks accounted for.

Conclusion

The project has been mostly successful, however needs revision to make realistic goals for the semester. The exponentiated gradient algorithm was easily ported over from MATLAB to Python, but the online lazy updates, due to its complexity, has been more difficult to debug and still poses issues. As a result, the project timeline needs to be revised, and can be done in a number of methods. However, the classroom portion is still on track, and thus the learning goals for the semester are unchanged.