Unit 3 Project - High-Performance Simulation Study (due 11-21)

Goal. Optimize your Unit 2 simulation study for computational efficiency and numerical stability. Apply some of the methods we have studied -- profiling, complexity theory, algorithmic improvements, array programming, and parallelization -- to reduce runtime.

Learning Objectives. By completing this project, you will:

- Profile code to identify computational bottlenecks
- Apply appropriate strategies to optimize your code

Project Requirements

Core Requirements

1. Document Baseline Performance

Before making any optimizations, document your Unit 2 simulation's baseline performance:

- **Runtime Profiling**: Use a profiler to identify where time is spent
- **Memory Profiling**: This step is optional
- **Computational Complexity Analysis**: Estimate how long a single simulation run takes as a function of your key parameter(s)
  - Example: If studying risk vs sample size n, time your code for various values of n
  - Assess computational complexity either by
    - Empirical analysis: e.g. fit a line on a log-log plot of runtime vs parameter value
    - Theoretically: an analysis of the main steps of your DGP(s), method(s), metric(s) as a function of the main parameter
- **Evidence of numerical instability**: Log any numerical warnings, convergence issues or exceptions raised during the execution of your simulation study that could be evidence of numerical instability
- **Baseline Metrics Document**: Create `docs/BASELINE.md` discussing:
  - Total runtime of entire simulation study (the version you submitted for Unit 2)
  - Results of a profiler (e.g. a table or a plot) summary showing main bottlenecks
  - Computational complexity of bottlenecks with theoretical and/or empirical evidence
  - Numerical warnings or convergence issues observed (document frequency and conditions)

2. Optimization Implementation

Implement strategies from **at least two** of the following categories:

Algorithmic Improvements:

- Replace inefficient algorithms with faster alternatives
- Eliminate redundant computations
- Use analytical solutions where possible instead of iterative methods

Array Programming:

- Vectorize operations to eliminate Python loops (even, potentially, loops over simulation replicates)
- Effectively leverage broadcasting

Numerical Stability:

- Identify and mitigate numerically unstable computations
- Implement stable algorithms for critical calculations (e.g., LogSumExp, stable variance)
- Add safeguards against overflow/underflow

Parallelization:

- Implement parallel simulation replicates
- Parallelize across simulation scenarios

Other approaches not enumerated here...

3. Optimization Documentation

Create `docs/OPTIMIZATION.md` that includes:

For each optimization implemented:

- **Problem identified**: What bottleneck or issue did profiling reveal?
- **Solution implemented**: What optimization strategy did you apply?
- **Code comparison**: Show before/after code snippets (can be brief)
- **Performance impact**: Runtime improvement, memory savings, or stability gains
- **Trade-offs**: Any costs (code complexity, readability, maintenance, precision)?

Profiling Evidence:

- Include profiler output or visualizations showing improvements
- Show flame graphs, timing comparisons, or memory plots if helpful

Lessons Learned: Answer any of the following:

- Which optimizations provided the best return on investment?
- What surprised you about where time was actually spent?
- Which optimizations were not worth the effort?

4. Updated Makefile

Extend your Makefile with new targets:

```
make profile         # Run profiling on representative simulation
make complexity      # Run computational complexity analysis (timing vs n)
make benchmark       # Run timing comparison: baseline vs optimized
make parallel        # Run optimized version with parallelization
make stability-check # Check for warnings/convergence issues across conditions
```

5. Performance Comparison Visualization

Create at least one visualization comparing baseline vs optimized performance:

- **Computational complexity plot**: Runtime vs key parameter (e.g., n) on log-log scale
  - Show both baseline and optimized versions
  - Include fitted lines or empirical complexity estimates
  - Consider plotting different components separately (DGP time, method time, etc.)
- **Overall timing comparison**: Show runtime improvements across different simulation scales or conditions
- **Speedup analysis**: Demonstrate scaling behavior (e.g., speedup vs number of cores for parallelization)

6. Regression Tests

Demonstrate that your optimizations preserve correctness:

- **Result validation**: Show that optimized code produces equivalent results to baseline. For example:
  - Use statistical tests comparing distributions of results
  - Check that summary statistics match within reasonable tolerance
  - Verify edge cases produce identical outputs
  - Create a test or script (`test_regression.py`) that verifies this.
- If your new code produces different results than your old code, explain why in `docs/OPTIMIZATION.md`.

7. Optional Extensions

Open-ended challenges for students who want to push further. Document your analysis in `docs/EXTENSIONS.md` if you choose to attempt one of these. You are also encouraged to explore some aspect of the efficiency/stability of your simulation study that I haven't mentioned,

and similarly document your findings in `docs/EXTENSIONS.md`.

Variance Reduction Techniques

Explore statistical techniques, such as Rao-Blackwellization or antithetic variates, to reduce the number of simulation replications needed to achieve a desired level of accuracy.

Simulation Budget Optimization

Consider the trade-off between computational cost and statistical precision. For different values of some parameter, e.g. n, the metrics of interest will have different variances. Statistically, this implies you may be able to use a lower value of n_{sim} for larger values of n. Computationally, this is advantageous, since large n scenarios are the slowest. Try to understand these relationships empirically and/or theoretically.

Pilot Study Design

Consider running a small-scale simulation study in order to get a sense of (1) how large n_{sim} should be or (2) what scenarios to prioritize. How would you design your pilot study? And how would the results inform your larger-scale study? Keep your main study aims in mind when navigating this.