Parallelized Retrospective Approximation

David Eckman (dje88)

Motivation

Simulation optimization deals with the problem

$$\min_{x \in \Theta} f(x),$$

where $\Theta \subseteq \mathbb{R}^d$ and f is a real-valued function that cannot be computed exactly, but instead can be estimated via stochastic simulation. We assume that $f(x) = \mathbb{E}[Y(x, \xi)]$ where Y is a deterministic real-valued function and ξ is a random element from a distribution that is not dependent on x. In sample average approximation (SAA), we fix a collection of random elements $\xi_1, \ldots, \xi_n$ and solve the problem

$$\min_{x \in \Theta} f_n(x) \quad \text{where} \quad f_n(x) = \frac{1}{n} \sum_{i=1}^{n} Y(x, \xi_i),$$

using some deterministic optimization algorithm. The minimizer of f_n, denoted x_n^*, is then taken to be an estimator of x^*, the minimizer of f.

A specialized version of SAA is retrospective approximation (RA) in which instead of solving one sample-path problem with large sample size, we solve a sequence of sample-path problems. More specifically, we solve a sequence of problems with increasing sample size m_k and decreasing error-tolerances ϵ_k. After some pre-specified number of iterations, we terminate the algorithm and take the final solution as our estimator of x^*.

Serial RA algorithm

1. Set the iteration number $k = 1$ and fix an initial solution x_0^*.
2. Generate a sample-path problem of size m_k. Use a deterministic optimization algorithm with a "warm start" at x_{k-1}^* to solve the generated problem to within an error-tolerance ϵ_k. Let x_k^* denote the solution.
3. Set $k \leftarrow k + 1$ and return to Step 2.

Idea

The question I wish to explore is: *What is gained if at each stage of the algorithm, we solve multiple sample-path problems in parallel and share their solutions?*

Here is an idea for a parallelized algorithm, running on p processors:

Parallel RA algorithm

1. Set the iteration number k = 1 and fix an initial solution x_0^*.
2. Generate p sample-path problems of size m_k and distribute the problems to the processors along with the previous solution x_{k-1}^*.
3. On each processor, use a deterministic optimization algorithm with a "warm start" at x_{k-1}^* to solve the generated problem to within an error-tolerance ϵ_k. Let x_k^j denote the solution where j is the index of the processor.
4. Aggregate the solutions $x_k^1, \ldots x_k^p$ on the master node and run a ranking and selection procedure to select the "best" solution, denoted x_k^*.
5. Set $k \leftarrow k + 1$ and return to Step 2.

I would like to analyze (empirically) how the quality of the solutions provided by the parallel algorithm compare to those of the serial algorithm, as a function of the iteration number and the wall-clock time.

References

Kim, S., Pasupathy, R., and Henderson., S. (2015). A Guide to Sample Average Approximation. In Fu, M. (Ed.), *Handbook of Simulation Optimization.* Springer, New York, 207–243.