We have conducted several tests using PDE pricer to see how our price and execution time depend on grid settings.

We used a put with K=2700, expiring in 2 years, constant volatility of 15%, S=2680 and interest rates from our source data, for as of date=2014-01-20.

## Performance and precision as a function of number of time points

Firstly, we used spot grid with 50 values and varied the number of time values between 10 and 1000 with step 10.

As expected, PV moves decrease as we increase the number of points

Execution time grew linearly:

## Performance and precision as a function of number of spot points

We also used time grid with 50 values and varied the number of spot values between 10 and 1000 with step 10.

We observed a much better convergence in terms of PV:

Performance graph has some random spikes, but in general is linear:

A different run produced different spikes, so we are sure this is some kind of environmental effect:

The graph’s general shape means, given our 3-diagonall matrix structure for solving equations, the solver is able to solve them with a linear complexity, not quadratic.

## Performance and precision as a function of number of time and spot points

As a combined experiment, we used time and spot grids with same numbers of points, varied values between 10 and 300 with step 1.

## Analysis

We see much more noise for big numbers of time points, compared to big numbers of spot points. The underlying reason is interpolation: our interest rates interpolation is piecewise-constant, while time is modelled as continuous, thus the result sometimes depends on which date the exact time point falls on. With spot we do not see such an effect as we do not have any unrealistic interpolation in spot space. We can observe decreased precision when we were varying number of time points, regardless if we were varying number of spot points or not. However, for a constant number of time points, we have a much better precision.

Calculation time is growing linearly in first two experiments and is (thus, predictably) growing quadratically in a combined experiment.

As for the performance aspect, we observe no spikes in calculation time when we are varying the number of time points only, but we do, when we are varying the number of post points. This makes us think NumPy linalg and/or python have some caches which allow them to efficiently deal with matrices of same size (when number of pot points is constant), but cause expensive operations (e.g., allocation, garbage collection, caching) when number of spot points changes.