IMPROVING CONVERGANCE ACCELERATION ALGORITHMS WITH MACHINE LEARNING

Project Report for the Data Mining and Machine Learning course [dsminingf17vm]

by Báskay János (NK.: UBCJY7)

2019.12.14.

Table of Contents

Motivation

The goal of this project is to improve the accuracy of the Aitken's delta-squared process on a Monte Carlo simulated sample of the CIR-phase diagram, by increasing the number of points with much smaller computational expense, than if those extra points were also generated by Monte Carlo simulations.

For this, one would think that maybe a multi-dimensional function fitting could give the right interpolation between the generated datapoints, however we will see that the connection between them is non-trivial, dependent on the discretization method, and has singular behavior near the critical point of the phase diagram, thus using machine learning might be advantageous.

Theoretical background

The Cox-Ingersoll-Ross-process

The Cox-Ingersoll-Ross-process is used in mathematical finance to describe interest rate movements driven by a single source of market risk. It's most common application is the valuation of interest rate derivatives.

The stochastic differential-equation that describes the model is the following:

$$dX_t = \kappa(\theta - X_t)dt + \sigma\sqrt{X_t}dW_t$$

Where W_t is a Wiener-process, and κ, θ, σ are parameters. The drift factor $\kappa(\theta - X_t)$ ensures mean reversion in the long run towards the value θ. The standard deviation factor $\sigma\sqrt{X_t}$ prevents the process from having negative values, so $X_t \in [0, \infty)$[1.]

Feller-condition and phase transitions

Introducing the new variable $\gamma = \frac{2\kappa\theta}{\sigma^2}$ the condition $\gamma \geq 1$ ensures that even $X_t = 0$ is forbidden.

Investigating the attainability of the lower boundary as a function of γ we will find three distinct phases:

$\boldsymbol{\gamma < 0}$	The process can't leave the 0-boundary	*"exit boundary"*
$\boldsymbol{0 < \gamma < 1}$	0-boundary is attainable and highly reflective	*"regular boundary"*
$\boldsymbol{\gamma \geq 1}$	0-boundary is unattainable	*"entrance boundary"*

Table 1: The phases of the CIR-process

Convergence acceleration

As we will later see numerical simulations of a stochastic process are always limited by the step size (dt) we choose, hence when we are measuring some phenomena eg. a phase diagram it will also be dependent on dt. The different curves belonging to different dt values (supposedly) has to converge to the curve belonging to "$dt = 0$", which we would obtain by analytically solving the same problem.

However, in practice it is advantageous to transform the original dt series, into a new one, that converges faster to the analytical limit. There are several series acceleration algorithms, now I'm going to use the Aitken's delta-squared process, where given a sequence $X = (x_n)_{n\in\mathbb{N}}$ a new sequence can be created:

$$(AX)_n = x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n}$$

Where

$$\Delta x_n = (x_{n+1} - x_n)$$

$$\Delta^2 x_n = x_n - 2x_{n+1} + x_{n+2}$$

[2.]

Data

The phase diagrams are obtained from the Monte Carlo trajectories of the CIR-Process by measuring the time needed for the trajectory to hit the 0-boundary (denoted by τ_0), and the "bulk", τ_{bulk} (by setting an other non-zero boundary, at 2ϵ if the process started from $X_0 = \epsilon$), and evaluating the probability $P(\tau_{bulk} > \tau_0) = P_{boundary}$ which is essentially the probability that the process can reach the 0-boundary in its lifetime. Then $P_{boundary}$ as a function of γ will yield the phase diagram.

Discretization schemes

When generating the Monte Carlo trajectories there are many ways to discretize the stochastic differential equation, some may be faster in computation time, but slower in convergence, others may have the opposite properties. I present here the ones I used for my simulations:

Euler-Maryama scheme:

$$X_{t+\Delta t} \approx X_t + \kappa(\theta - X_t)\Delta t + \sigma\sqrt{X_t}W_t\sqrt{\Delta t} \qquad W_t \sim \mathcal{N}(0,1)$$

Weighted Milstein Scheme

$$X_{t+\Delta t} \approx \frac{X_t + \kappa(\theta - \alpha X_t)\Delta t + \sigma\sqrt{X_t}W_t\sqrt{\Delta t} + \frac{1}{4}\sigma^2\Delta t(W_t^2 - 1)}{1 + (1-\alpha)\kappa\Delta t}$$

Where $0 \leq \alpha \leq 1$ is the weight.

Balanced Implicit Scheme

$$X_{t+\Delta t} \approx \frac{X_t\big(1 + C(t, X_t)\big) + \kappa(\theta - X_t)\Delta t + \sigma\sqrt{X_t}W_t\sqrt{\Delta t}}{1 + C(t, X_t)}$$

Where

$$C(t, X_t) = \kappa\Delta t + \frac{\sigma\sqrt{\Delta t}|W_t|}{\sqrt{X_t}}$$

Pathwise Adapted Linearization Quadratic scheme

$$X_{t+\Delta t} \approx X_t + \big(\kappa(\tilde{\theta} - X_t) + \sigma\beta_t\sqrt{X_t}\big)\left(1 + \frac{\sigma\beta_t - 2\kappa\sqrt{X_t}}{4\sqrt{X_t}}\Delta t\right)\Delta t$$

Where $\beta_t = \frac{W_t}{\sqrt{\Delta t}}$, and $\tilde{\theta} = \theta - \frac{\sigma^2}{4\kappa}$

Most of my examples will use data generated with the Euler-Maryama scheme, but I will also evaluate the optimized Machine-Learning model on data generated by other discretization schemes.

Other optimizations

To reduce the computation time for the simulations, I parallelized the trajectory generation process, and since both τ_0 and τ_{bulk} is much smaller than the full runtime of the simulation (T), we can introduce an early stopping method, once the trajectory hits any of the barriers specified above.

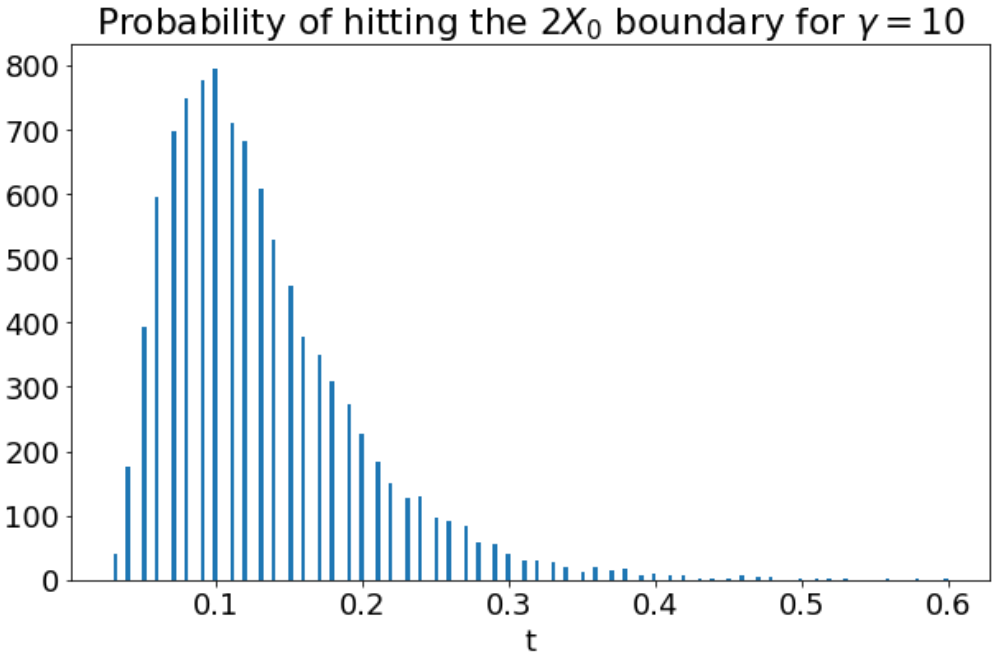

Figure 1:The time it takes to breach the barrier is less than 1, while the simulation ran through T = 100

Exploring the dataset

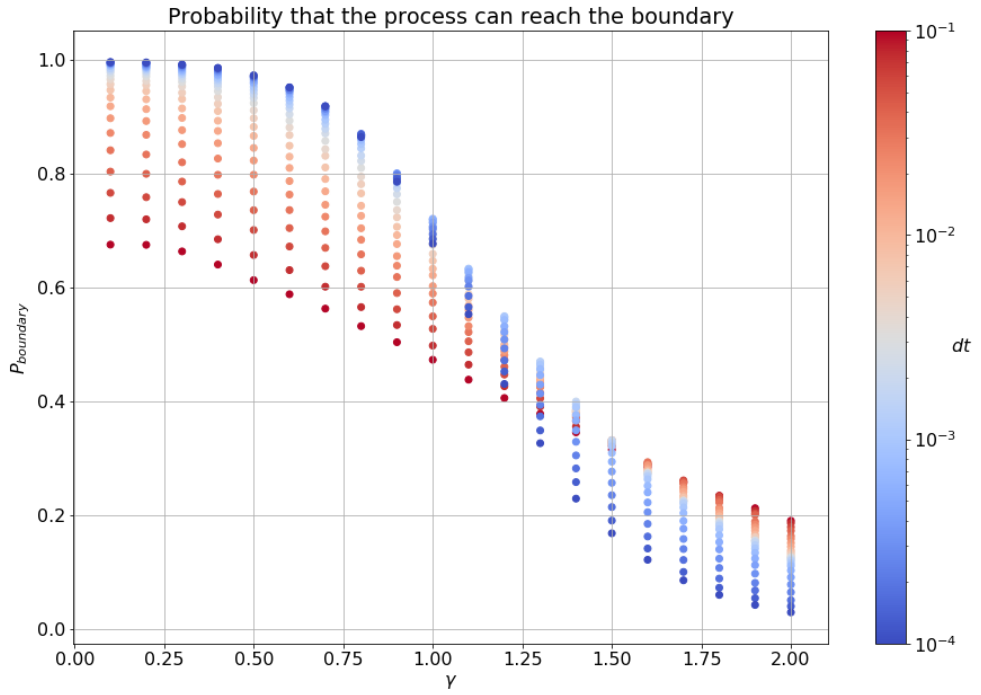

Figure 2: Phase diagram for many dt values

Figure 2. shows the phase diagrams generated by simulations, that I will use to train and validate my model. It is clear that the $\gamma < 1$ part converges to 1, while the other goes to 0.

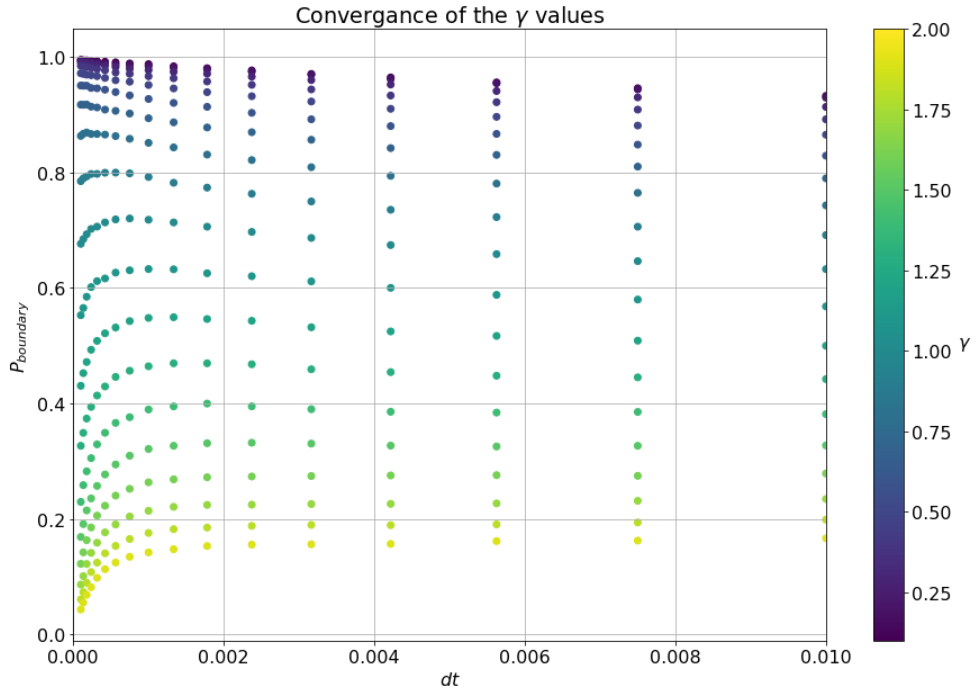

Figure 3: dt series for different γ-s

On *Figure 3.* We can see, that the convergence is non-linear, so methods for accelerating linear series are unusable, and we can even expect that even the non-linear methods will perform unoptimally.

Model selection

To select the optimal machine-learning algorithm for the task, I will use leave-one-out-cross validation to test each of them with their default settings, using the mean absolute error as a metric. I tested the following models: *Linear Regression*, *K-nearest neighbors regression*, *Random Forest regression*, *XGBoost* (both the tree-like and forest-like versions) and sklearn's *Multi-layer-Perceptron regressor*.

The results can be seen on *Figure 4.*. Based on that I used, and fine-tuned *XGBoost*'s Regressor.

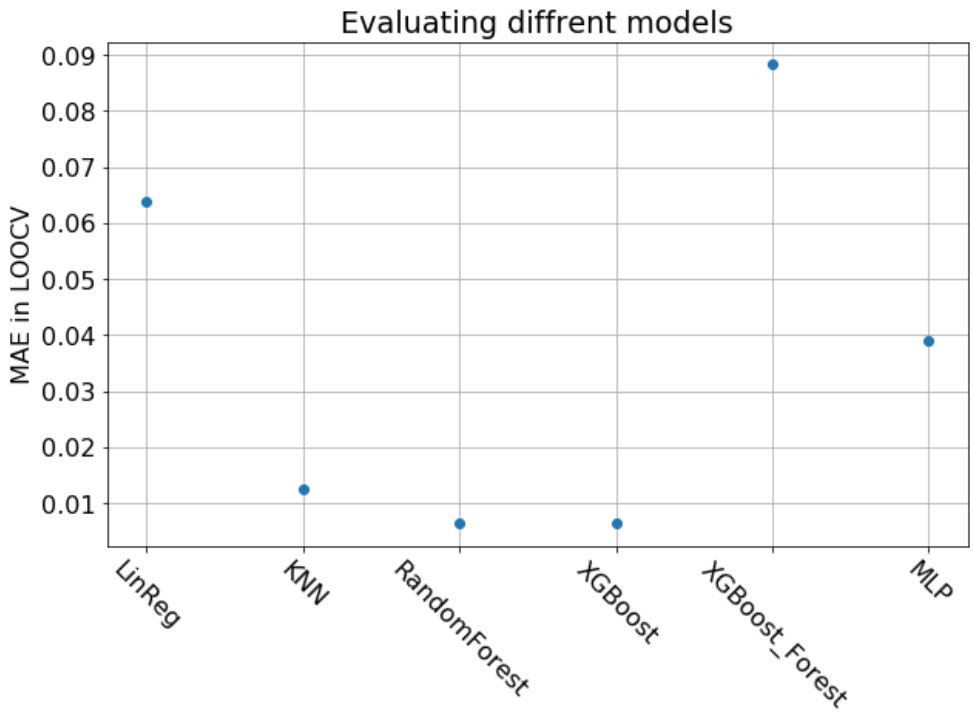

Figure 4: Performance of different models

Hyperparameter tuning

There are two parameters, that can have a large influence on the performance of *XGBoost*, the maximum depth of trees (*max_depth*) and the number of trees to fit (*n_estimators*). *Figures 5.* and *6.* Show the results.

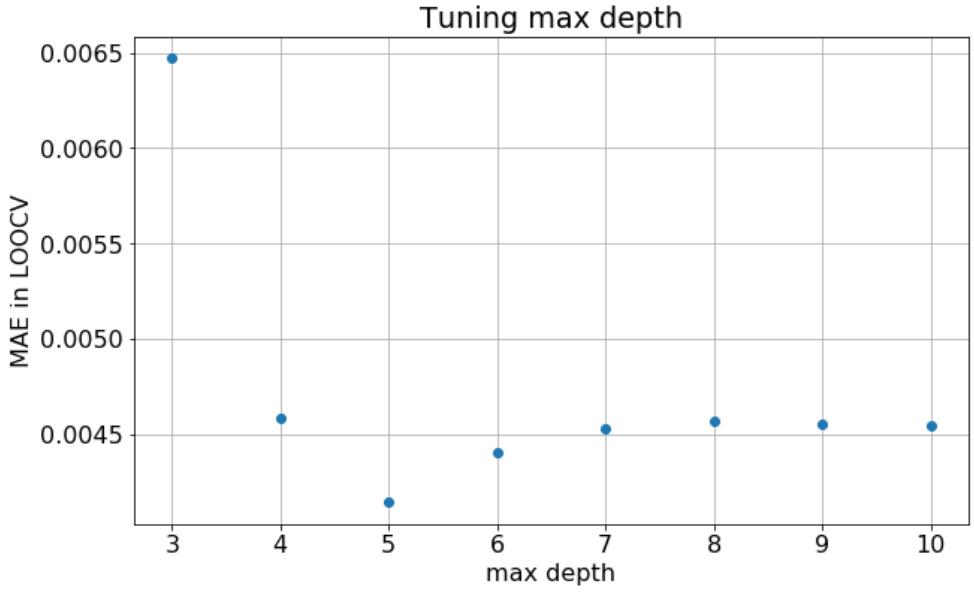

Figure 5: Finding the optimal depth of trees

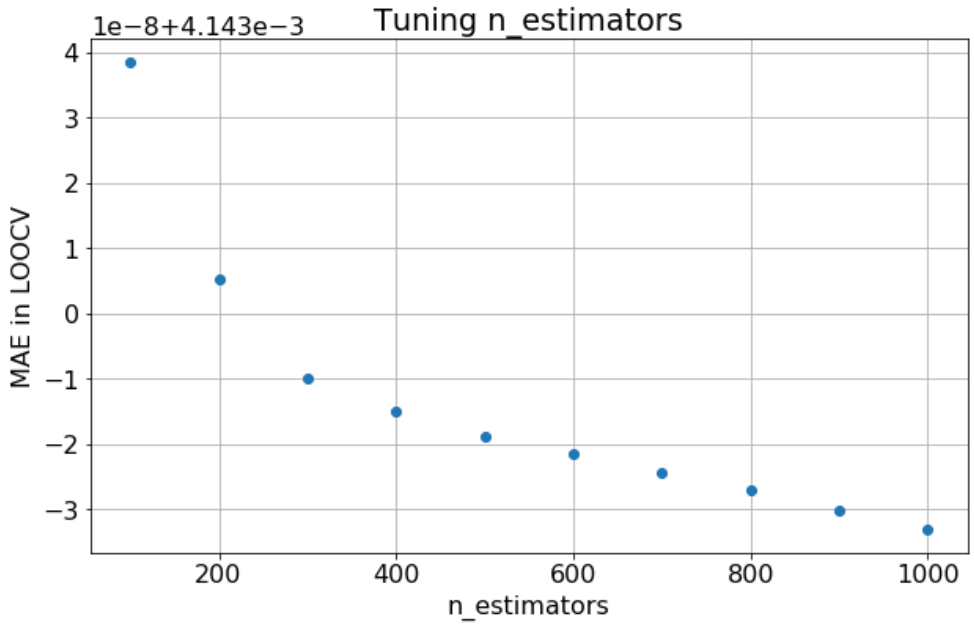

Figure 6: Finding the optimal number of trees

It is clear that the depth of *5* is optimal, but for the number of trees, the result is the more, the better, so there we have to consider the run time of the fitting, and choose something that's fast enough, without compromising too much on the accuracy, that is why I choose *500*.

With these optimizations the MAE of the model went from *0.0065* to *0.0041*. This is an order of magnitude smaller, than the smallest value the model has to predict.

Testing on other discretization methods

Since this model was fitted for data generated with the Euler-Maryama shceme it is interesting to see, how it performs on the others. Table 2. contains the MAE in LOOCV for all of them.

Discretization scheme	MAE in LOOCV
Euler-Maryama	0.0041
Weighted Milstein	0.00099
Balanced Implicit	0.00056
Pathwise Adapted Linearization Quadratic	0.0043

Table 2: Model's performance on different schemes

This model works well for any discretization schemes, which was one of the goals of this project.

Predicting new values with the optimal model

Using *XGBoost* with a *max_depth* of *5* and a *n_estimators* of *500* I predicted new phase diagrams within the region spanned by the known points for 100 gamma values and a 100 dt-s. If I were to obtain this from Monte Carlo simulations, I'd have to run it for approximately a week, the predictions were ready in milliseconds.

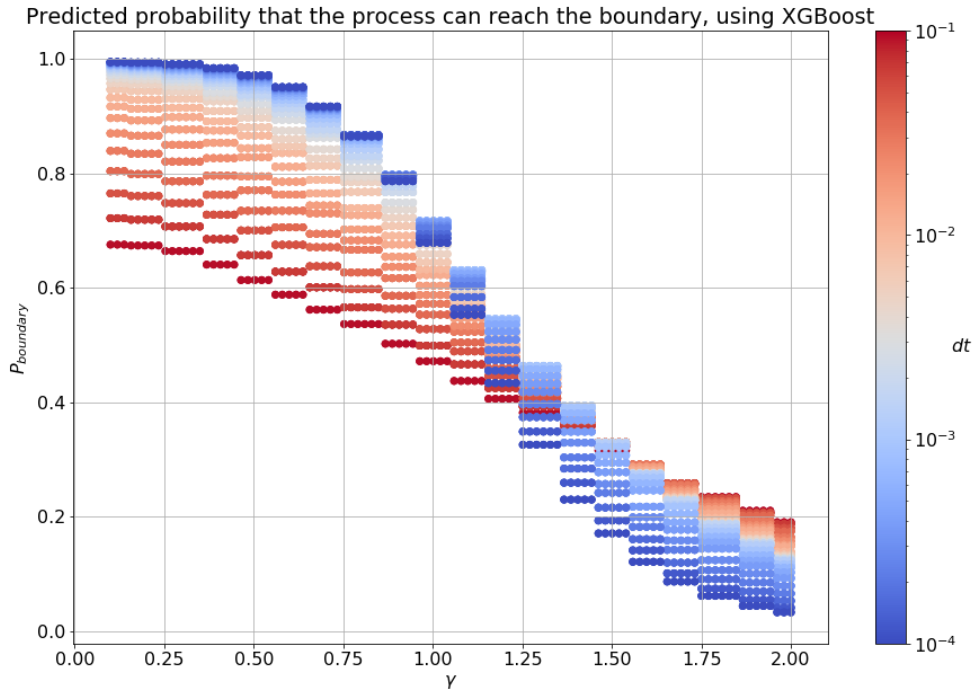

Figure 7: Predicted phase diagram for many dt values

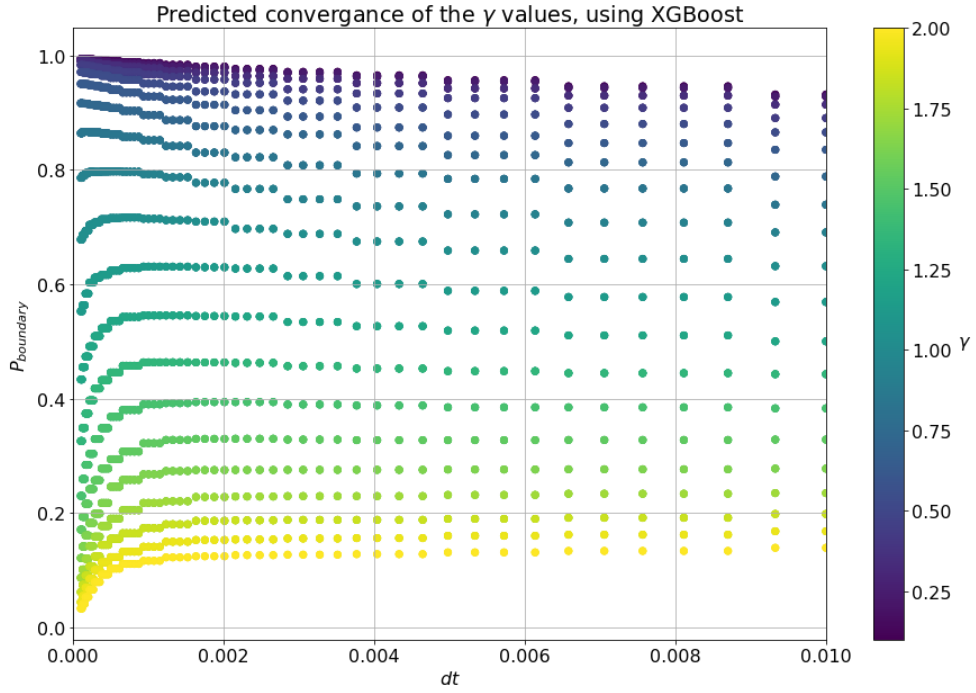

Figure 8:Predicted dt series for different γ-s

Figures 7. and *8.* show clear signs of overfitting, all the predicted values are the known values for a given range. Essentially this model quintupled every known point in the γ direction and quadrupled them in dt.

Conclusion

While I didn't expect to create a model, that just multiples the existing points by a factor determined by the number of points to predict, I decided to see how the Aitken's delta-squared process changes when using the simulated and the predicted dataset.

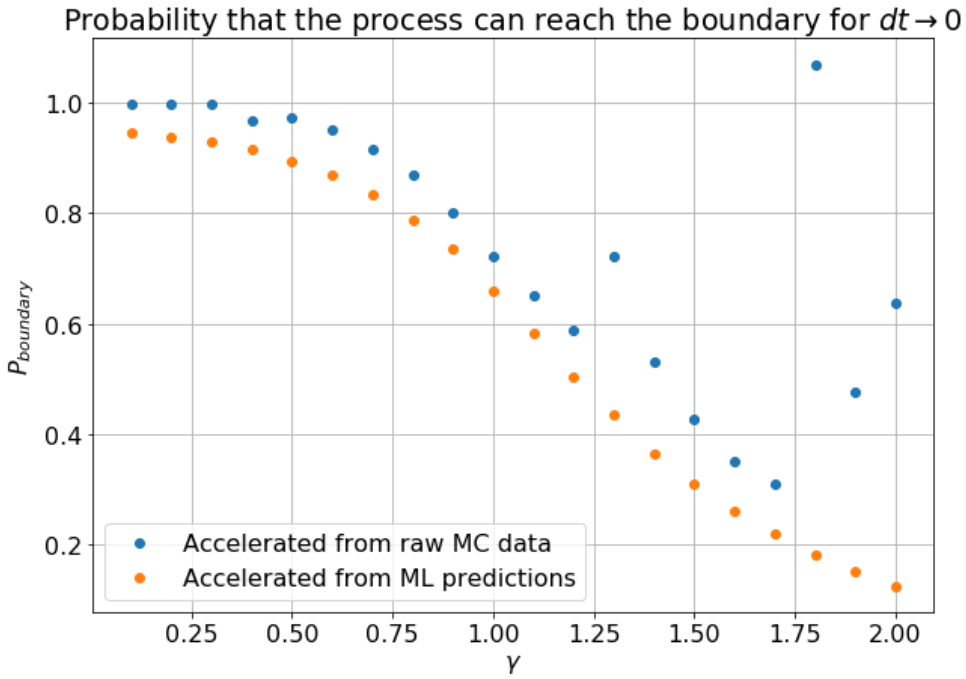

Figure 9: Phase diagram after series acceleration

Figure 9. shows an improvement in the $\gamma > 1$ regime, but the curve made from the predictions has consistently lower values than the simulation data.

Overall, I don't think Machine-Learning techniques can improve series acceleration methods by much, but this project showed, that multiplying the known points (with a little noise on top) can indeed improve the Aitken's delta-squared process and smoothes out some imperfections, caused by the small dataset.

References

[1.] Hull, John C. (2003*). Options, Futures and Other Derivatives*. Upper Saddle River, NJ: Prentice Hall. **ISBN 0-13-009056-5**

[2.] Alexander Aitken, "*On Bernoulli's numerical solution of algebraic equations", Proceedings of the Royal Society of Edinburgh (1926)* **46** pp. 289–305.

The code (at least some version…) used for simulation can be found here: https://github.com/baskayj/Boundary-behaviour-in-stochastic-differential-equations-used-in-Finance