

Gaussian Process Regressions for Equity Option Risk Metrics - A Cautionary Tale

Rodney Greene, Ph.D.

The aim of this little note is to urge my fellow quant researchers to take great care when using Gaussian Process Regression (GPR) as a proxy risk calculation framework. Quants are often faced with the business requirement to calculate risk metrics under baseline and stressed scenarios for hundreds of thousands (or millions?) of derivative positions. Since the production trading models may require too much time and memory to handle this task, a pre-trained GPR grid is a flexible and rationale proxy-framework¹. Nevertheless, I take the simple case of calculating equity option Δ to show the results need to be carefully assessed for business use adequacy.

Specifically I describe three ways to estimate option Δ , the partial derivative of option value with respect to the underlying stock price, ($\Delta = \frac{\partial \text{optionValue}}{\partial \text{underlyingStockPrice}}$)

- The standard finite-difference method
- The GPR-analytic method
- The *pyTorch automatic differentiation method*

The methods obtain roughly the same result and generally speaking, they are all equally bad for the single call option test case described in this study.

The note begins with an overview of the GPR methodology with specific mappings to the *GPyTorch* Python package. I assume the reader is familiar with the Black-Scholes formula for call options. To create my call option value training data I use the *quantLib* Python package. Following the GPR overview I include the results where I qualitatively compare the exact Black-Scholes Δ to the three methods above. My goal is to use this simple call option example to create a sense of cautious optimism in the quant community regarding GPR-based risk estimates for large derivative portfolios. All the code may be sourced from the *blackScholesDeltaGPR* GitHub repository.

1 Gaussian Process Regression Primer

The classic reference for Gaussian Processes is Ref [7]. Consider a scalar-valued process driven by a time-dependent vector, $f(\mathbf{x})$, corrupted by additive i.i.d. noise, ϵ . This noise is usually modeled as $\mathcal{N}(0, \sigma_n^2)$. The n observations of this corrupted signal are $\{y_i \in \mathbf{y}(\mathbf{x}), \mathbf{x}_i \in \mathbf{X}, i \in [1, \dots, n]\}$. The GP *ansatz* states the prior distribution for these observations is a zero-mean multivariate Gaussian, $p(\mathbf{y}(\mathbf{x})) = \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}))$. The covariance of the observations $\mathbf{y}(\mathbf{X})$ is written as a kernel matrix-valued function $\mathbf{K}(\mathbf{X}, \mathbf{X})$. An essential component of GPR engineering is selection of a suitable kernel. In this study I only use the *squared exponential* (also called the *radial basis function*) kernel.

¹See Ref. [1] for an excellent description of GPR-based risk and CVA metrics for interest rate swaps.

The regression problem is to estimate the predictive posterior distribution of the uncorrupted process (evaluated at test points, $\mathbf{f}_*(\mathbf{X}_*)$) conditional on the observations of the corrupted signal, $\mathbf{y}(\mathbf{X})$. The first step is to apply the GP *ansatz* to write the GP prior for the observations and test points. We obtain

$$\begin{bmatrix} \mathbf{y}(\mathbf{X}) \\ \mathbf{f}_*(\mathbf{X}_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} & \mathbf{K}(\mathbf{X}, \mathbf{X}_*) \\ \mathbf{K}(\mathbf{X}_*, \mathbf{X}) & \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right) \quad (1)$$

Upon applying classical statistical conditioning and marginalization ² (specifically, conditioning on the observations) one obtains the *predictive posterior* as

$$p(\mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}_*) = \mathcal{N}(\langle \mathbf{f}_* \rangle, \mathbf{cov}(\mathbf{f}_*)), \text{ where} \quad (2)$$

$$\langle \mathbf{f}_* \rangle = \mathbf{K}(\mathbf{X}_*, \mathbf{X}) [\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{y}(\mathbf{X}) \quad (3)$$

$$\mathbf{cov}(\mathbf{f}_*) = \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X}) [\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{K}(\mathbf{X}, \mathbf{X}_*) \quad (4)$$

The *GPyTorch* Python package is a GPR implementation that supports GPU acceleration, which is useful for calculating the computationally intensive matrix inversions in the calculation of the predictive posterior. The *gPyTorch.models* class is a representation of the GP prior, $p(\mathbf{f}(\mathbf{x}))$. It supports a generalization to non-zero mean, although that flexibility is rarely employed in practice. The GP prior is constructed with a user-specified covariance kernel function.

The class *gpytorch.likelihoods* is a representation of the predictive posterior, *i.e.*,

$$\text{likelihood}(\text{model}(\mathbf{X})) = p(\mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}_*).$$

2 GPR-Based Equity Option Δ

I employ a simple laboratory to test out GPR-based risk metrics - Δ for a 30-Day european-exercise style call option struck at 100 on a single underlying stock. This laboratory is useful because one may calculate a closed-form exact Black-Scholes Δ to use as a benchmark for the GPR-based Δ . The standard way to numerically estimate Δ is as a centered finite difference, $\Delta \simeq \frac{\text{value}(s+\delta s) - \text{value}(s-\delta s)}{2\delta s}$. The values are regressed on the value-trained Gaussian Process data structure. The so-called *GPR-analytic* Δ estimate is given as Eq (10) in Ref. [1]³.

$$\Delta_{\text{GPR-analytic}} = \frac{\mathbf{k}(\mathbf{x}_{\text{test}}, \mathbf{x}_{\text{train}})}{\partial \mathbf{x}_{\text{test}}} \left[\mathbf{K}(\mathbf{x}_{\text{train}}, \mathbf{x}_{\text{train}}) + \sigma_{\text{noise}}^2 \mathbf{I} \right]^{-1} \mathbf{y}_{\text{train}} \quad (5)$$

The leading partial derivative of the covariance kernel, $\frac{\mathbf{k}(\mathbf{x}_{\text{test}}, \mathbf{x}_{\text{train}})}{\partial \mathbf{x}_{\text{test}}}$ is an indication the GPR-based Δ is clearly dependent on the kernel choice. In order to calculate $\Delta_{\text{GPR-analytic}}$ my code in the GitHub repository makes heavy use of the *pyTorch* broadcast framework for

²See Appendix A.2 of Ref. [7].

³See Section 2.7 of Ref. [4] for a derivation of the derivative of Gaussian Process values.

tensor operations. Finally, pyTorch implements *Automatic Differentiation* which calculates the GPR-based Δ recursively via backward traversal of the Directed Acyclic Computation Graph that is used for the Δ calculation.. Ref. [5] discusses the pyTorch automatic differentiation API whereas Ref. [3] is a theoretical description of this powerful technique.

Figure 1 shows the result of the GPR regression of the call option value. The training data is polluted with additive noise to easily model bid/ask spread around the theoretical Black-Scholes value. Qualitatively I think the regression looks good. The stochastic optimization used for the hyperparameter optimization used 100,000 iterations. I observed the results dramatically improves as the number of iterations increases. The take-away from this figure is when training a GP with option values, the resulting regression of the values is satisfactory.

The picture is not so rosey when we compare the GPR-based Δ estimates to the exact Black-Scholes Δ - see Fig 2. The GPR trained per Fig 1 is used to calculate the finite-difference, GPR-analytic and automatic differentiation GPR-based Δ estimates. It is apparent they all compare poorly to the exact Black-Scholes Δ . As expected, I observed these results dramatically improved as I increased the number of stochastic optimization iterations in the hyperparameter tuning step from 100 to 100K. I used the squared exponential kernel for the results shown in the figures.

3 Conclusion


I've shown that GPR-based european option Δ calculation must be done carefully. The laboratory test cases explored herein compares poorly to the exact Black-Scholes Δ . Despite the appealing flexibility of Gaussian Process Regression as a proxy for CPU/memory-intensive derivative risk calculations, quants should go down this route after careful study of the quality of the results with respect to hyperparameter optimization and choice of the covariance kernel.

4 Side Note on Matrix Inversion

It's useful to say a few words about the calculation of Eq. (5) for the GPR-analytic Δ calculation. Since the matrix that is being inverted is positive definite, it is usual to employ Cholesky factorization to calculate the matrix inverse. The Cholesky factor of a positive definite matrix \mathbf{A} is a lower-triangular matrix such that $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. So we may define the term $\boldsymbol{\lambda}$ as follows

$$\begin{aligned}\boldsymbol{\lambda} &= \left[\underbrace{\mathbf{K}(\mathbf{x}_{train}, \mathbf{x}_{train}) + \sigma_{noise}^2 \mathbf{I}}_{\mathbf{L}\mathbf{L}^T} \right]^{-1} \mathbf{y}_{train} \\ &= \mathbf{L}^{-T} \mathbf{L}^{-1} \mathbf{y}_{train}\end{aligned}$$

RLG Technologies[©]



Consider the last two terms in this equation.

$$\begin{aligned}\mathbf{L}^{-1}\mathbf{y}_{train} &\equiv \mathbf{z} \Rightarrow \\ \mathbf{L}\mathbf{z} &= \mathbf{y}_{train}\end{aligned}$$

Because this is a lower-triangular system of equations it can be solved robustly and quickly with `torch.triangular_solve`. So now we may plug this into our λ equation to obtain

$$\begin{aligned}\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{y}_{train} &= \mathbf{L}^{-T}\mathbf{z} \equiv \mathbf{g} \Rightarrow \\ \mathbf{L}^T\mathbf{g} &= \mathbf{z}\end{aligned}$$

In similar fashion, we can robustly and quickly solve this triangular system for \mathbf{g} with `torch.triangular_solve`. The results is

$$\begin{aligned}\lambda &= (\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{y}_{train} \\ &= \mathbf{g}\end{aligned}$$

In the actual code I don't have to call `torch.triangular_solve` twice - calling `torch.cholesky_solve` once is equivalent.

References

- [1] Crépey, Stéphane, Matthew F. Dixon, *Gaussian Process Regression for Derivative Portfolio Modeling and Application to CVA Computations*.
- [2] *GPyTorch*
- [3] Griewank, Andreas, *On Automatic Differentiation*, Rice University preprint, 1989.
- [4] McHutchon, Andrew, *Nonlinear Modelling and Control using Gaussian Processes*, Ph.D. Thesis, Cambridge University, 2014.
- [5] *A GENTLE INTRODUCTION TO TORCH.AUTOGRA*D
- [6] *BROADCASTING SEMANTICS*
- [7] Rasmussen, Carl E. and Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*, MIT Press.

Figure Captions

Figure 1. Theoretical Black-Scholes european exercise style call option value (30-Day expiry). Also shown is the training data (with additive noise) and the resulting confidence band from the Gaussian Process regression. Here the stochastic hyperparameter optimization employed 100K iterations. It is apparent the quality of the GPR regression is satisfactory.

Figure 2. Black-Scholes call option Δ , as calculated per the theoretical formula, finite difference, GPR-analytic formula Eq. (5) , and by using pyTorch automatic differentiation. Note the three estimates are all similarly unsatisfactory.

Figures

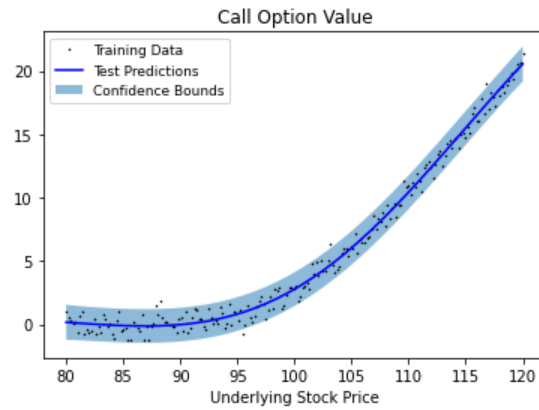


Figure 1: Theoretical Black-Scholes european exercise style call option value (30-Day expiry). Also shown is the training data (with additive noise) and the resulting confidence band from the Gaussian Process regression. Here the stochastic hyperparameter optimization employed 100K iterations. It is apparent the quality of the GPR regression is satisfactory.

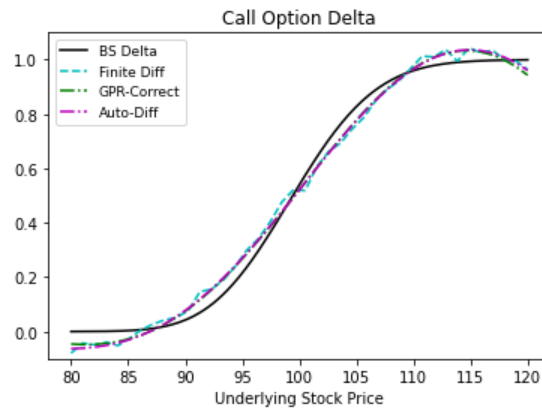


Figure 2: Black-Scholes call option Δ , as calculated per the theoretical formula, finite difference, GPR-analytic formula Eq. (5), and by using pyTorch automatic differentiation. Note the three estimates are all similarly unsatisfactory.