ETH zürich

Gaussian process regression methods for option pricing

Nicholas Delmotte

Outline

Introduction

This thesis/presentation is divided into three parts.

In a first part, the theory of Gaussian process regression is explained in the context of pricing European call options. This part serves as a quick recap of a previous presentation from ~6 months ago.

In the second part, a first potential extension of GPR is explored: the pricing of any European option/payoff function. This is achieved by replacing the usual scalar inputs (T, K) of a model by a functional input f which represents the payoff function of the option to price. In this part, the question of No Arbitrage conditions is also considered.

In the third and last part, another extension is studied: the use of multi-output Gaussian processes to simultaneously price American and European put options. The idea here is to associate two outputs $(P_{\text{Am}}, P_{\text{Eur}})$ to the usual inputs (T, K), and hopefully the correlation between the two outputs will yield better estimates than if the two option types were priced separately.

Outline

Definition

A Gaussian process (GP) is a stochastic process of a set of random variables such that any finite subset of these random variables follows a multivariate normal distribution.

$$(X_{t_1}, \dots, X_{t_n}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

$$\boldsymbol{\mu} = \text{mean vector}, \Sigma = \text{covariance matrix}$$

We are interested in using Gaussian Process Regression to learn the pricing functional $\hat{P}$.
Given a data set $\mathcal{D} = \{(T_i, K_i, Y_i)\}_{i=1}^{N}$, we learn the pricing map $\hat{P}$ that approximates the option prices Y_i:

$$(T, K) \mapsto \hat{P}(T, K)$$

Gaussian process regression

The true structure of our data set is such that for each tuple $\mathbf{x}^i = (T_i, K_i)$ we have a noisy price observation $Y_i \simeq P(T_i, K_i)$, where $P(\cdot)$ is the *true* price. We then try to learn the price surface $\hat{P}(\mathbf{x})$ based on a Gaussian noise observation model:

$$Y(\mathbf{x}) = P(\mathbf{x}) + \epsilon(\mathbf{x}), \quad \epsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

We make the assumption that the map $\mathbf{x} \mapsto P(\mathbf{x})$ is a realisation of a Gaussian random field, such that any subset $\{P(\mathbf{x}), \mathbf{x} \in \mathcal{X}\}$ is a multivariate Gaussian:

$$(P(\mathbf{x}^1), \dots, P(\mathbf{x}^n)) \sim \mathcal{N}(\mathbf{m}_n, \mathbf{K}_n)$$

where for n data points we identify the prior mean vector $\mathbf{m_n}$ and covariance matrix K_n:

$$\begin{aligned} \mathbf{m}_n &= (m(\mathbf{x}^1; \boldsymbol{\beta}), \dots, m(\mathbf{x}^n; \boldsymbol{\beta})) \\ \mathbf{K}_{ij} &= \kappa(\mathbf{x}^i, \mathbf{x}^j; \boldsymbol{\beta}) \end{aligned}$$

Here, $\kappa(\cdot, \cdot)$ denotes a positive semi-definite kernel function, and $\boldsymbol{\beta}$ is the vector of all hyperparameters.

Gaussian process regression

The aim of GPR is to estimate the posterior distribution of $P(\cdot)$ given the observations data $\mathcal{D}$ and assuming a Gaussian noise model. One observes that the conditional posterior predictive distribution $P(\mathbf{x}_*)|\{\mathbf{x}_i, y_i\}_{i=1}^n$ is also Gaussian, and its mean yields the GPR estimation of the price, $\hat{P}(\mathbf{x}_*)$. We define this pointwise mean as m_*:

$$\begin{aligned} m_*(\mathbf{x}_*) &= \mathbb{E}[P(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}] \\ &= m(\mathbf{x}_*) + K^T(\mathbf{K}_n + \sigma_\epsilon^2\mathbf{I})^{-1}(\mathbf{y} - \mathbf{m}_n) \end{aligned}$$

In the above equation we identify $\mathbf{x}_*$ as an arbitrary input, $\mathbf{X} = [\mathbf{x}^1, \dots, \mathbf{x}^n]^T$ and $K^T = [\kappa(\mathbf{x}_*, \mathbf{x}^1; \boldsymbol{\beta}), \dots, \kappa(\mathbf{x}_*\mathbf{x}^n; \boldsymbol{\beta})]$. Finally, the covariance of the posterior distribution on a chosen set of inputs $\{\mathbf{x}_*^i\}_{i=1}^M$ is given by:

$$(K_*)_{ij} \equiv \text{Cov}(P(\mathbf{x}_*^i), P(\mathbf{x}_*^j)) = \kappa(\mathbf{x}_*^i, \mathbf{x}_*^j; \boldsymbol{\beta}) - K_i^T(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}K_j$$

These two quantities provide the two key ingredients to our GPR approach: $m_*(\mathbf{x}_*)$ is the most likely estimate of $P(\mathbf{x}_*)$, whereas Var$(P(\mathbf{x}_*))$ indicates the model uncertainty.

Determining the hyperparameters

Gaussian Process Regression relies on determining appropriate values for the hyperparameters $\boldsymbol{\beta}$ that shape the posterior distribution. We find these hyperparameters in the prior kernel function, the prior mean function and the noise estimate (also known as the 'nugget').

- Kernel function: choices include Gaussian (RBF) or Matérn (5/2,3/2)

$$\kappa_{RBF}(\mathbf{x},\mathbf{x}') = \sigma_p^2 \exp\left(-\sum_{k=1}^{d} \frac{(\mathbf{x}_k - \mathbf{x}'_k)^2}{2 l_{\text{len},k}^2}\right)$$

$$\kappa_{M52}(\mathbf{x},\mathbf{x}') = \sigma_p^2 \prod_{k=1}^{d} \left(1 + \frac{\sqrt{5}}{l_{\text{len},k}}|\mathbf{x}_k - \mathbf{x}'_k| + \frac{5}{3 l_{\text{len},k}^2}(\mathbf{x}_k - \mathbf{x}'_k)^2\right) e^{-\frac{\sqrt{5}}{l_{\text{len},k}}|\mathbf{x}_k - \mathbf{x}'_k|}$$

- Mean function: constant, linear, or polynomial

$$m(\mathbf{x};\boldsymbol{\beta}) = \sum_{k=1}^{K} \beta_k \phi(\mathbf{x}), \quad \phi(\cdot) \text{ a polynomial basis}$$

- Observation noise:

$$\epsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

Determining the hyperparameters

The total set of hyperparameters to be trained is thus:

$$\{\{\beta_i\}_{i=1}^{K}, \sigma_p^2, \{l_{\text{len},k}\}_{k=1}^{d}, \sigma_\epsilon^2\}$$

Maximum likelihood estimation is used to find the optimal parameters for the model. The likelihood function of the parameters β given the data $\mathbf{y}$ is given by the multivariate Gaussian probability distribution $f(\mathbf{y}|\beta, \mathbf{x})$:

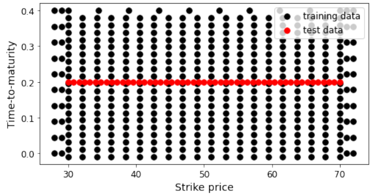
$$\begin{aligned}\mathcal{L}(\beta|\mathbf{y}) &= f(\mathbf{y}|\beta, \mathbf{x}) \\ &= \frac{1}{2\pi\sqrt{|K^{n\times n} + \sigma_\epsilon^2 \mathbb{1}|}} \exp[-\frac{1}{2}(\mathbf{y} - \mathbf{m}^n)^T (K^{n\times n} + \sigma_\epsilon^2 \mathbb{1})^{-1} (\mathbf{y} - \mathbf{m}^n)]\end{aligned}$$

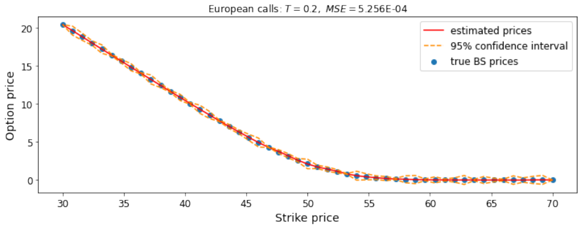
The optimal hyperparameters $\beta^\star$ are then found by the optimisation problem:

$$\beta^\star = \arg\max_{\beta} \mathcal{L}(\beta|\mathbf{y}) = \arg\max_{\beta} f(\mathbf{y}|\beta, \mathbf{x})$$

Implementation and results

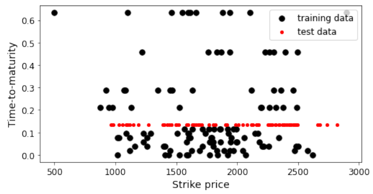
We generate a grid of (T, K) input points and compute the associated Black-Scholes call prices. We use this data to train the model, and then test it on a new test data set with varying K and fixed $T = 0.2$.

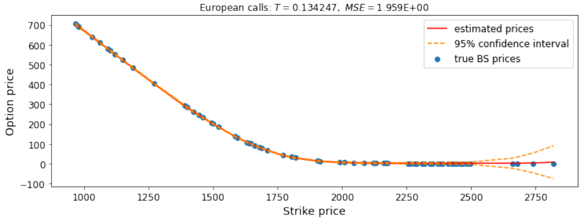




Implementation and results

Let’s also try using real options data instead of generating Black-Scholes prices.





Outline

Building a reference point with the Finite Element Method

Before we look at implementing a GP, let's briefly consider the FEM which we'll use as a reference point for our GP.

We approximate a function $f(x)$ by a linear combination of basis functions $\phi_i(x)$: $u(x) = \sum_i \gamma_i \phi_i(x)$.
We choose hat functions as our basis since they obey a few convenient properties: $\phi_i(x_j) = \delta_{i,j}$ for equally spaced $x_i = ih$, and $\phi_i(x)$ is 0 when x is not close to x_i.
We require that our approximation $u(x) = \sum_i \gamma_i \phi_i(x)$ passes through some select points, namely: $u(x_i) = f(x_i)$. We obtain that $\gamma_i = f(x_i)$.

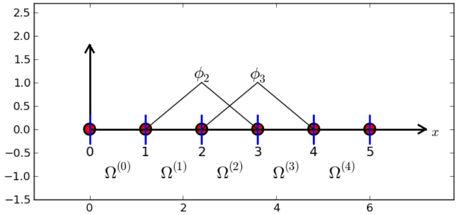


Illustration of basis hat functions [1].

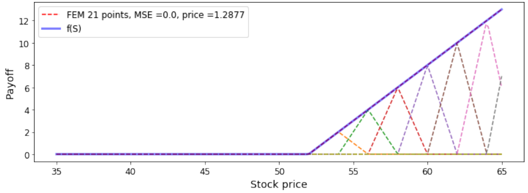
Building a reference point with the Finite Element Method

In a financial context, Butterfly options are the equivalent to hat functions, and can be constructed from 3 call options: $B(K) = C(K-a) - 2C(K) + C(K+a)$. We can approximate any payoff function with the Butterfly basis and compute the corresponding option price as:

$$f(x) \approx \sum_i \gamma_i B_i(x)$$

$$P = \mathop{\mathbb{E}}_{x\in\Omega}(f(x)) \approx \mathop{\mathbb{E}}_{x\in\Omega}\Big(\sum_i \gamma_i B_i(x)\Big)$$

$$= \sum_i \gamma_i \mathop{\mathbb{E}}_{x\in\Omega}(B_i(x))$$



Approximation of a call function of strike $K = 52$. The real price is $P = 1.2994$.

Functional Gaussian processes
Theory

Having presented a simple first attempt using the finite element method, we now try to solve the same pricing problem using Gaussian process regression.
We wish to approximate a functional g which we represent as a Functional Gaussian Process of zero mean and covariance operator k: $g(v) \sim \mathcal{FGP}(0, k(v, v')), \quad v, v' \in V$. Here V is a space of input functions to our functional.
We choose our covariance operator to be the integral: $k(v, v') = \theta \int_{\Omega} v(x)v'(x)dx$.
Similarly to the classical Gaussian Process case, we can define a regression procedure given training data $\mathbf{\Phi} = [\phi_1, \phi_2, ..., \phi_M]$ with corresponding observed outputs $\mathbf{y}$, and test data $\mathbf{\Phi}^* = [\phi_1^*, \phi_2^*, ..., \phi_{M^*}^*]$.
We can write the posterior distribution $\mathbf{g}^*|\mathbf{y}, \mathbf{\Phi}, \mathbf{\Phi}^* \sim \mathcal{N}(\bar{\mathbf{g}}^*, \text{cov}(\mathbf{g}^*))$, where:

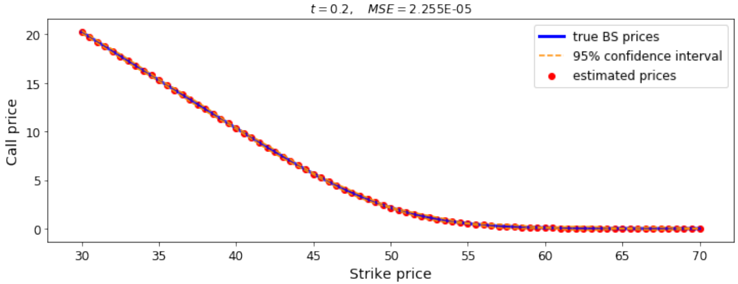
$$\bar{\mathbf{g}}^* = \mathbf{K}(\mathbf{\Phi}^*, \mathbf{\Phi})(\mathbf{K}(\mathbf{\Phi}, \mathbf{\Phi}) + \sigma^2\mathbf{I})^{-1}\mathbf{y}$$

$$\text{cov}(\mathbf{g}^*) = \mathbf{K}(\mathbf{\Phi}^*, \mathbf{\Phi}^*) - \mathbf{K}(\mathbf{\Phi}^*, \mathbf{\Phi})(\mathbf{K}(\mathbf{\Phi}, \mathbf{\Phi}) + \sigma^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{\Phi}, \mathbf{\Phi}^*)$$

Functional Gaussian processes

Results: comparing to the FEM

To compare the FEM and the FGP model, we choose to train our model on Butterfly functions: these are the input to our model. The output is the corresponding Black-Scholes price of the input payoff function.

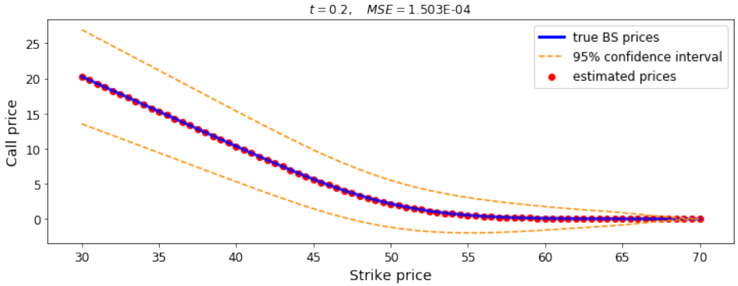


This figure shows the FGP approximation of call prices for varying strikes. The finite element method yields a call price curve estimate with a mean squared error of 2.176×10^{-3}, about two orders of magnitude worse than the Gaussian process estimate.

Functional Gaussian processes

Results: The FGP model cannot extrapolate

It is interesting to note that similarly to the FEM, the FGP model cannot extrapolate. In the figure below, the FGP model was trained on a slightly narrower range of Butterfly functions (no longer on the $K \in [30, 70]$ range). The result is a worse MSE, and completely non-sensical confidence bounds.



Functional Gaussian processes

Results: Extending to other payoff functions

Having tested the FGP model against call prices, it would be interesting to see how well it can price a larger variety of payoff functions. The table below shows the performance of the FGP model when pricing different types of payoff functions. An FEM comparison is also given.

Payoff function	FGP estimated price	FGP estimated std dev.	FEM estimated price	True price
$f(x) = \exp\left(-\frac{(x-50)^2}{2*10^2}\right)$	0.88851	0.0033543	0.87917	0.88868
$f(x) = x^2$	2543.8	9.0252	2541.5	2542.8
$f(x) = \cos(x) + 1$	1.0202	0.33957	1.0044	0.99172
$f(x) = \begin{cases} 0 & \text{if } x < 50 \\ 1 & \text{if } x \geq 50 \end{cases}$	0.51057	0.067152	0.66241	0.50854

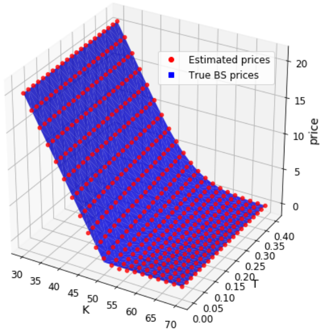
Functional Gaussian processes

Extending to two dimensions

Extending the model to include a time dimension is made relatively straightforward by the fact that multiplying two valid kernels yields another valid kernel. We can thus replace our previous kernel by:

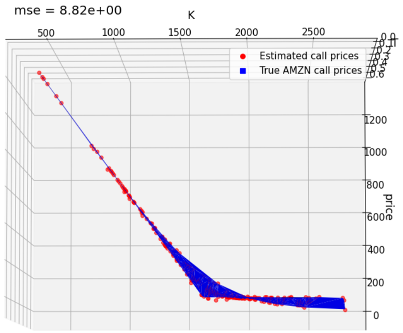
$$k(v, v', T, T') = \theta \int_{\Omega} v(x)v'(x)dx \cdot \exp\left(-(T - T')^2/(2 * l_T^2)\right)$$

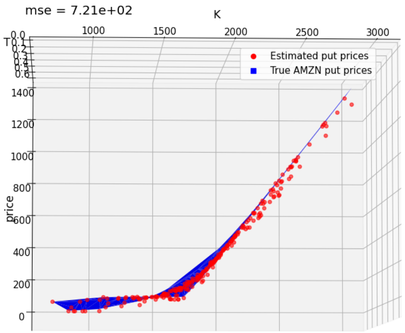
The figure on the right shows the resulting call price surface.



Functional Gaussian processes
Using real data

We can try to plot similar surfaces, this time using real data to train and test our model. The figures below show the resulting call price and put price surfaces (the model was only trained on call prices).





Implementing the No Arbitrage conditions

We shall focus only on the two following conditions:

1. The price of a call option is convex in strike.
2. The price of a call option is monotone increasing in time-to-maturity.

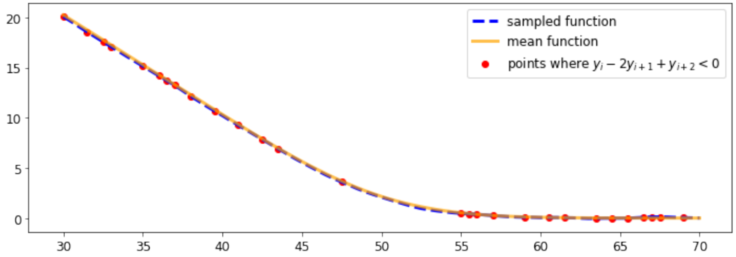
Three different methods for implementing these conditions were considered:

1. Rejecting non-convex (or non-monotone) samples
2. Markov chain Monte Carlo methods
3. Projecting the non-convex (or non-monotone) samples onto convex (monotone) curves

Implementing the No Arbitrage conditions

Rejecting non-convex (or non-monotone) samples

The first idea to enforce the No Arbitrage conditions on our model is to simply reject the samples yielded by the model that break the conditions, and keep those that satisfy them. The figure below shows a random sample from our posterior distribution, and the points where it is non-convex. Clearly, given the number of such points, if we simply run the sampler until we get a satisfactory sample we'll be waiting a long time...



Implementing the No Arbitrage conditions
MCMC methods

Another idea is to use Markov Chain Monte Carlo methods to generate convex, monotone samples. Let us consider the two ‘basic’ algorithms.

Metropolis-Hastings algorithm

This algorithm starts with a sample and generates another sample dependent directly on the first one. Since, at each step, we are generating a completely new call price curve, this method is akin to the simple rejection method described above, and would take a similarly large number of steps to produce even one completely convex, monotone sample.

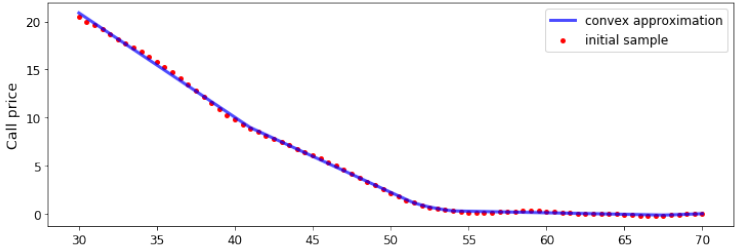
Gibbs sampling

In this method, instead of generating the whole sample/vector in one go, each step consists in changing the value of only one element at a time. In this way, instead of the whole n points needing to satisfy the conditions, we only need one point to satisfy them at each step. The problem with this method is that it only really works if the initial sample is already convex, which defeats the point. Also, after a short test run it became apparent that the new samples tended to remain almost or exactly equal to the initial convex sample.

Implementing the No Arbitrage conditions

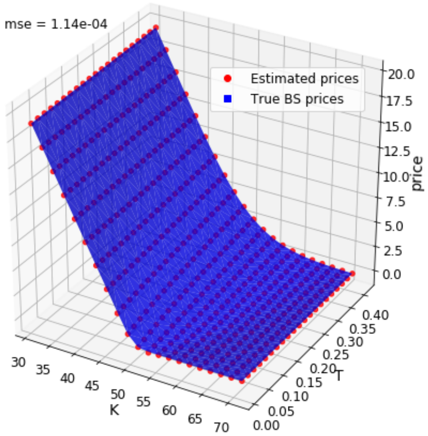
Projecting onto convex, monotone samples

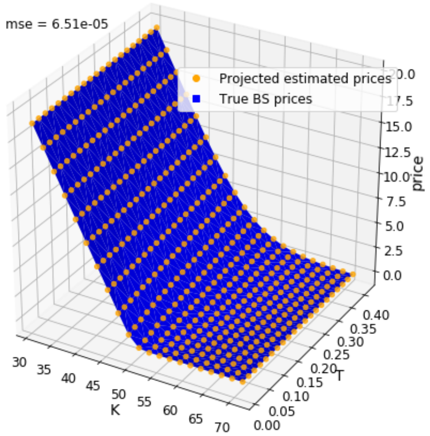
An alternative to sampling convex samples is to simply project the non-convex samples onto the set of convex curves, in order to obtain their nearest convex approximation. The problem becomes: given a non-convex curve f, find the curve f' that minimises $|f - f'|$ such that $f'(K-h) - 2f'(K) + f'(K+h) \geq 0 \quad \forall K, h$. This problem can be solved easily in Python. The figure below shows an (exaggerated) example of such a projection of a non-convex sample onto its nearest convex approximation.



Implementing the No Arbitrage conditions
Projecting onto convex, monotone samples

We can also extend the conditions to 2D.





Outline

Generating option prices with the binomial tree model

Since we can't price American options with the B-S model, we turn to the binomial tree model which can actually be seen as a discrete time approx. of the B-S model. The principle of the model is that, at each time step, the price of the underlying stock S_0 can either go up to a value uS_0 with probability p or down to a value dS_0 with probability $1-p$. We choose $d = 1/u$ which yields the final price of the stock:

$$S_n = S_0 u^{N_u - N_d}$$

The payoff at the final time step is simply:

$$C_n = (K - S_n)_+ = (K - S_0 u^{N_u - N_d})_+$$

The price at the penultimate time step can then be computed as:

$$C_{n-1} = e^{-rT/n} \left(pC_n(\text{up}) + (1-p)C_n(\text{down})\right)$$

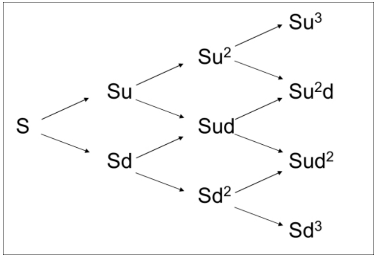


Illustration of the binomial tree pricing model [2].

Multi-output GPs
The Intrinsic Coregionalisation Model

Consider two output functions $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ which we wish to approximate using a multiple output Gaussian process. Suppose we have Gaussian process $u(\mathbf{x}) \sim \text{GP}(0, k(\mathbf{x}, \mathbf{x}'))$ which we can sample twice to obtain $u^1(\mathbf{x})$ and $u^2(\mathbf{x})$. We can now approximate f_1 and f_2 by adding a scaled transformation of u^1 and u^2:

$$f_1(\mathbf{x}) = a_1^1 u^1(\mathbf{x}) + a_1^2 u^2(\mathbf{x})$$
$$f_2(\mathbf{x}) = a_2^1 u^1(\mathbf{x}) + a_2^2 u^2(\mathbf{x})$$

By writing $\mathbf{a^1} = [a_1^1, a_2^1]^T$, $\mathbf{a^2} = [a_1^2, a_2^2]^T$, and $\mathbf{f} = [f_1, f_2]^T$, we have:

$$\begin{aligned}\text{cov}(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}')) &= \mathbf{a}^1(\mathbf{a}^1)^T \text{cov}(u^1(\mathbf{x}), u^1(\mathbf{x}')) + \mathbf{a}^2(\mathbf{a}^2)^T \text{cov}(u^2(\mathbf{x}), u^2(\mathbf{x}')) \\ &= \mathbf{a}^1(\mathbf{a}^1)^T k(\mathbf{x}, \mathbf{x}') + \mathbf{a}^2(\mathbf{a}^2)^T k(\mathbf{x}, \mathbf{x}') \\ &= [\mathbf{a}^1(\mathbf{a}^1)^T + \mathbf{a}^2(\mathbf{a}^2)^T] k(\mathbf{x}, \mathbf{x}')\end{aligned}$$

We define $K^f = \mathbf{a}^1(\mathbf{a}^1)^T + \mathbf{a}^2(\mathbf{a}^2)^T$, and obtain:

$$\text{cov}(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}')) = K^f k(\mathbf{x}, \mathbf{x}') = \begin{pmatrix} K_{11}^f & K_{12}^f \\ K_{21}^f & K_{22}^f \end{pmatrix} k(\mathbf{x}, \mathbf{x}')$$

Multi-output GPs
The Intrinsic Coregionalisation Model

In fact, using the Kronecker product $\otimes$, we can write:

$$\begin{bmatrix}\mathbf{f}_1\\ \mathbf{f}_2\end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}_1)\\ \vdots \\ f_1(\mathbf{x}_N)\\ f_2(\mathbf{x}_1)\\ \vdots \\ f_2(\mathbf{x}_N)\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{0}^N\\ \mathbf{0}^N\end{bmatrix}, K^f \otimes k^x\right)$$

where K^f is a positive semi-definite matrix that specifies the inter-task similarities.
Given the expression in the equation above, the posterior mean prediction of the model at a test point $\mathbf{x}_*$ is given by:

$$\mathbf{f}_*(\mathbf{x}_*) = (K^f \otimes \mathbf{k}_*{}^x)^T \Lambda^{-1}\mathbf{y}, \qquad \Lambda = K^f \otimes K^x + \Sigma \otimes I$$

Multi-output GPs

Using noiseless data at the same data locations for each task

A paper by Bonilla et al. [3] underlines an issue surrounding the use of the ICM. The problem is that, if we consider noiseless data, then using the same input data set for both outputs yields no inter-task transfer. The proof is succinct:

Proof.

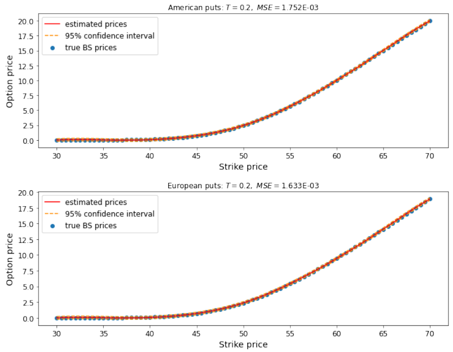
In the noiseless case $\Sigma = 0$ and so we have:

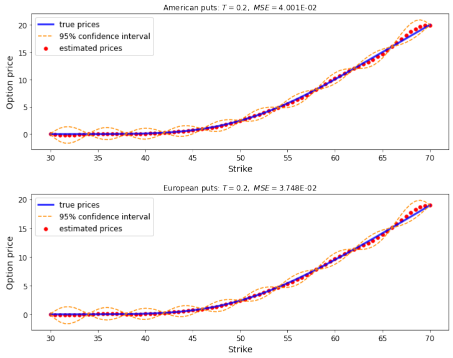
$$\begin{aligned}
\mathbf{f}_*(\mathbf{x}_*) &= (K^f \otimes \mathbf{k}_*{}^x)^T (K^f \otimes K^x)^{-1}\mathbf{y} \\
&= ((K^f)^T \otimes (\mathbf{k}_*{}^x)^T)((K^f)^{-1} \otimes (K^x)^{-1})\mathbf{y} \\
&= [(K^f)^T (K^f)^{-1} \otimes (\mathbf{k}_*{}^x)^T (K^x)^{-1}]\mathbf{y} \\
&= \begin{pmatrix} (\mathbf{k}_*{}^x)^T (K^x)^{-1}\mathbf{y}_1 \\ \vdots \\ (\mathbf{k}_*{}^x)^T (K^x)^{-1}\mathbf{y}_D \end{pmatrix}
\end{aligned}$$

Thus, in the noiseless case, the predictions for a task d depend only on the targets $\mathbf{y}_d$. □

Multi-output GPs
Using noiseless data at the same data locations for each task

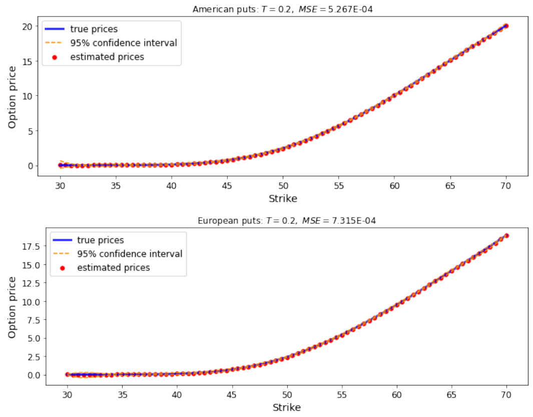
The following figures illustrates the problem. On the left are the predictions of two separately trained GPs. On the right are the predictions of a multi-output GP model trained simultaneously on American and European data.





Multi-output GPs
Using different data sets for each task

To justify the use of a multi-output GP we can then either use very noisy data, or we can use different training data sets for each output. The intuition is that, if we train the model using a grid of 100 (T, K) points associated to American puts and a separate grid of 100 (T', K') points associated to European puts, then the model might learn something about American puts at the (T', K') points and likewise for European puts at the (T, K) points. The figure on the right shows the results of such an implementation.



Outline

Conclusion

- We showed the versatility of Gaussian process regression, which can be adapted to fit various scenarios.
- Our first extension was a partial success; we were able to slightly improve on the performance of the FEM.
- We were able to implement NA conditions in 2 dimensions, and obtained better results after the conditions were applied.
- We demonstrated the potential of multiple output GPs in pricing different option types simultaneously.

Acknowledgements

- Thank you to my colleague Daniel Montagna for aiding me in the first part of this thesis.
- Thank you to Wahid, Jakob, Hanna, and Philippe for supervising the thesis.
- Thank you to Prof. Teichmann for allowing me to do my thesis in his group, and for supervising as well.

Other works and possible improvements

- Sparse variational GPs use inducing points to reduce the complexity of the regression algorithm from $O(n^3)$ to $O(nm^2)$, where $m \ll n$, n the number of training points and m the number of inducing points.

- Deep Gaussian processes are a natural extension of the multi-output GPs we've seen.

- A fully-Bayesian approach could be explored, as shown in the paper by M. Tegnér [4].

Bibliography

- H. P. Langtangen, “Introduction to finite element methods.” http://hplgit.github.io/INF5620/doc/pub/main_fem.pdf, Dec. 2013. Accessed: 23/03/2021.
- quant.stackexchange user16651, “Time 0 value of an american put in cox-ross-rubinstein model.” https://quant.stackexchange.com/questions/17268/time-0-value-of-an-american-put-in-cox-ross-rubinstein-model. Accessed: 13/04/2021.
- E. Bonilla, K. Chai, and C. Williams, “Multi-task gaussian process prediction.,” 01 2007.
- M. Tegnér and S. Roberts, “A probabilistic approach to nonparametric local volatility,” 2019.