Neural Network for Volatility Surfaces under Convex Constraint

Tristan Cazenave [*] **Carlo Sala** [†‡] **Timothée Sohm-Quéron** [§]

January 9, 2020

Abstract

This paper proposes a new data-driven machine learning-based approach to estimate the volatility surface and infer the conditional Risk-Neutral Density (RND) from a cross-section of daily option prices. In order to retrieve the RND from option prices, one need to insure, among others, that the price of the options is convex with respect to the strike price. By designing an optimal design of the neural network, our paper proposes two new approaches to guarantee that the convexity constraint is always respected. The two proposed approaches follow the same economic principles but use a different number of layers in estimations. On top of the convexity constraints, the estimated measures are arbitrage-free, time-varying, computationally simple and fast to retrieve. To empirically demonstrate the properties of our approach, we recover the implied volatility and RND for the S&P500 options over a period of almost 20 years.

Keywords: Risk-Neutral Density, Volatility Surface, Convex Constraints, Neural Network
JEL classification: C13; C14; G13.

[*]Paris-Dauphine University, LAMSADE, Paris E-mail: cazenave@lamsade.dauphine.fr.

[†]Department of Financial Management and Control, ESADE Business School, Ramon LLull University, Avenida de Torreblanca 59, 08172 Sant Cugat, Barcelona, Spain; E-mail: carlo.sala@esade.edu.

[‡]Financial support from the AGAUR - SGR 2017-640 grant is gratefully acknowledged.

[§]Paris-Dauphine University, LAMSADE, Paris and Bramham Gardens E-mail: timothee@bramham-gardens.com.

1 Introduction

The most accessible data, such as price history and accounting data, used in quantitative finance applications tend to be backward-looking. Daily stock prices are not per-se backward-looking, but their data structure make them not suitable to infer a density. At each point in time, a stock price is in fact just a single scalar, out of which is not possible to infer a density. A common way out to obtain a density is to smooth an histogram of historical returns. Unfortunately, the resulting density would be backward-looking and highly dependent on the amount of historical returns used for the estimation. As such, even in the presence of a fully transparent and efficient stock market the recovered density would be biased being only partially informative. Differently, options market data give us a way to have some insight about how investors see the future returns. Forward-looking densities can be naturally be inferred from option market data, thanks to their richer data structure. At each point in time options market data have a matrix structure, with different states of the world (strike prices) and time horizons (time-to-maturity). Predictions are usually summarized into the Risk-Neutral Density (RND) which, in turns, summarizes the underlying option cross-section, for a particular time horizons. Interestingly, the option cross-section encodes the investors' beliefs about the value of the underlying at different time horizons and states of the world. A large literature focuses on how these expectations are able to predict different events, such as financial crises (Birru and Figlewski (2012), Bates (1991) or Christoffersen et al. (2013)). While the theory underpinning the estimation of the RND is well-known since Breeden and Litzenberger (1978), there are several practical issues that makes the estimation of the RND not straightforward (details follow). In this paper we propose a new arbitrage-free, machine-learning and data-based approach that allows for the inference of the RND from the daily option cross-section. As a main advantage with respect to the existing literature, the proposed approach is computationally fast, adaptive and does not violate the options convexity with respect to the strike prices, a key feature to produce an economically valid RND.

Breeden and Litzenberger (1978) show that it is possible to extract the RND in presence of a continuum of strike prices. Specifically, at each time and for each time-to-maturity, the current RND relative to a future time T of a given financial asset S_T, can be inferred from the option cross-section by taking the second derivative of the European option price - e.g. a European call option, C, with respect to the strike price, K (see appendix for the derivation):

$$f(S_T) = e^{rT} \frac{\partial^2 C}{\partial K^2} \tag{1}$$

where r is the today value of a risk-free asset such that e^{rT} accounts for the time value of money. This procedure gives a model-free estimates of the RND linked to the strike price used in estimation. The main issue is that, even for heavily traded markets such as the S&P 500, there is rarely a continuum of strike prices, and some kind of interpolation is needed. In addition of being sparse, option market data are often also noisy. A smooth interpolation is then not always possible, above all into the tails of the distribution, where data are scarcer and more prone to mispricing. Aside from being numerically tractable, and possibly even more importantly, any models for the estimation of the RND has to obey basic no-arbitrage principles. Such principles require the strict positivity for the RND and different shape constraints, e.g. the convexity constraints. Finally, the obtained outputs must provide an accurate fit, both in- and out-of-sample. In this paper we propose a novel approach that produces very good results, both in- and out-of-sample and satisfies the no-arbitrage conditions.

Understanding the volatility surface is a key objective for both practitioners and academics in the field of finance. Given its stochastic nature, fitting the volatility surface - which is the collection of the implied volatilities of all strike prices for different maturities - is not an easy task and, to date, there is still no definitive consensus on the approach to use. It is a known fact, confirmed by many different empirical investigations that the volatility surface has a number of dynamic and static stylized facts. Generally, the volatility surface is steeper for short maturity options, negatively skewed (again more strongly for short-date options and during bad market times), mean reverting, and displays e negative correlation with the underlying returns.[1] In addition of capturing these complex behaviour, any model for the volatility surface must be guided by no-arbitrage principles.

For ease of convenience, the many models present in the literature can all be classified in two

[1] Rebonato (2013), among many others, a more comprehensive analysis of the properties of the volatility surface.

broad categories, parametric and non-parametric models. Parametric models try to fit the parameters of the underlying process, and get the implied volatility using market data. As common for all parametric models, these approaches are defined in closed form, fast and not data intensive. Unfortunately, parametric models typically rely on unrealistic assumptions which are hardly confirmed by market. Lacking of the necessary flexibility to fit real market data the obtained final results are often misleading. Non-parametric models directly model the implied volatility without any explicit assumption about the market dynamics of the underlying. While more data intensive, the obtained results are usually better approximation of the true values. Among many others, Skiadopulos (2001) and Christian (2008) provide a large literature survey of the many different techniques that exist in the literature to fit the volatility surface. From a more practical perspective Fengler (2011) shows the importance of having a smooth and stable volatility surface for pricing and hedging exercises.

The same parametric/nonparametric classification can also be applied for the different models present in the literature aiming to estimate the RND. Jackwerth (1999), Bondarenko (2003) and Birru and Figlewski (2012) provide a comprehensive literature review of the topic and conclude that, as of today, no methods is clearly superior. The parametric approaches can be further classified in i) models that directly estimate the RND, ii) models that first invert the estimated volatility smile and then estimate the RND following Breeden and Litzenberger (1978), and iii) models for the stochastic process. Among the parametric models for the RND there are i) the expansion approaches of Jarrow and Rudd (1982), Madan and Milne (1994), Longstaff (2015) which approximate the RND by correcting known distributions; ii) the mixture model approaches of Ritchey (1990), and Giacomini et al. (2008) which approximate the RND by mixing different parametric distributions; iii) the generalized distribution approach of Rosenberg (1998) which approximate the RND by interpolating flexible multivariate RND inferred by different asset prices with the aim of adding additional parameters that may capture possibly key features, e.g.: the higher moments of the distribution. Among the models for the volatility smile Shimko (1993), Malz (1997), Birru and Figlewski (2012) and Laurini (2011) first fit implied volatilities using functional forms of different kind, and then follows Breeden and Litzenberger (1978) to extract the RND from the fitted prices. Fi-

nally, among the models for the stochastic process we have Black and Scholes (1973), Hull and White (1987), Heston (1993), Bates (1996). As for the volatility surface parametric approaches also the RND parametric models produce parsimonious and tractable but often misspecified final outputs.
The nonparametric approaches can be further classified in i) Implied trees, ii) Smoothing techniques, iii) Maximum entropy iv) Neural networks (details follow). The implied trees model of Derman and Kani (1994), Dupire (1994) and Rubinstein (1994) infer the risk-neutral probabilities from observed European option prices and use them to infer a consistent and recombining unique binomial tree. Smoothing techniques can be applied for the estimation of the RND, e.g.: Jackwerth and Rubinstein (1996); for the estimation of the volatility surface, e.g.: Campa et al. (1998)[2], Bliss and Panigirtzoglou (2004) and Jackwerth (2000); or with different kernel approaches which outputs are then used to estimate the RND following Breeden and Litzenberger (1978) as in Äit-Sahalia and Lo (1998) and Äit-Sahalia and Duarte (2003). The maximum entropy approaches of Stutzer (1996) and Buchen and Kelly (1996) minimize the cross-entropy to a prior distribution to find the optimal RND given observations.

As common for all nonparametric approaches, these models produce a good fit with real market data but are very unstable and produce unreliable final output if fed with small samples. Along the same line, and common to all cited models that rely on Equation (1) for the estimation of the RND, the estimation is numerically very delicate, being prone to the curse of differentiation. Being the RND computed as a ratio, small errors due to data sparsity or noise are easily propagated and amplified possibly leading to very biased and unstable final outcomes.

For 25 years, machine learning proposed interesting alternatives to the above approaches, mainly using neural networks (see e.g. Malliaris and Salchenberger (1993) or Andreou et al. (2008) and reference therein). Some papers try to infer the parameters of the dynamics of the underlying for the sake of rapidity, such as in Pironneau (2019) who fits the parameters of the Heston model using a neural network. Mostafa et al. (2015) propose a way to better taking into account the different characteristics of the option cross-section (e.g.: options moneyness) by partitioning the volatility surface and then building a particular neural network model for

[2]More precisely, the paper of Campa et al. (1998) compares 3 smoothing techniques with an implied binomial tree technique and a mixture of lognormal distributions on a novel dataset of OTC options.

each specific characteristic. Other authors achieve apparently impressive results, but the outcomes are often difficult to interpret economically, being heavily guided by the "black-box" characteristic of neural networks, e.g.: Yu Zheng and Chen (2019). Good performances have also been achieved through "less" common machine learning approaches, e.g. the random forests of Crespo (2018). Once more, the improved performances arising from using more complex approaches usually comes at a price, as these methods might not to be easy to used by practitioners, being too "black-box".
Overall while the finance approaches are usually economically rigorous but often of impractical use whenever the underlying cross-section of option prices is not liquid enough, the machine learning approaches usually provides good quality results, but without having a robust economic interpretation. In this paper we will join the two stream of literature proposing an economically-grounded, machine learning approach to extract the volatility surface and the RND. Ludwig (2015) also provides economically-grounded machine learning approach, but with different economic constraints.

The remainder of our paper is organized as follows. Section 2 presents more formally the problems encountered in the machine learning literature for the estimation of the RND. Section 3 review the basic elements of the neural network technology. Section 4 show how to include the convexity constraints into the neural network approach.

2 Problem

We see major issues in the new methods. First, most of the applications of neural networks (whatever its deepness) only take two inputs to fit the volatility surface: the maturity and the level of moneyness. This means that the output of the network is extremely static from one day to the next and does not take into account what happens in markets, as the change in predictions can only be obtained through a re-estimation of the network on another time window. Second, the use of different networks to direct different regions of the volatility surface means that the networks are not connected with one another. Obviously, it would lead to an improvement in terms of fit, but would also introduce major discontinuities in terms of second derivative of the option prices with respect to the strike.

Generally, the target assigned to the neural network (see next part) is to minimize the error between actual option prices and prices predicted by the network. This is not enough as it does not emphasize the connectivity between adjacent points. Yu Zheng and Chen (2019) provided an elegant way to take into account the non-arbitrage conditions, generating pseudo-data from the network and penalizing the loss if non-arbitrage conditions are verified. However, this approach might be very long and not suitable for practical use (taking one month to backtest 5116 trading days on a single PC, even with parallel programming on a GPU), and still has the issue of the limited number of inputs. In the meantime, traditional approaches also struggle. Derived methods suffer from the need to specify a particular dynamics for the underlying asset. Direct methods (such as splines with a convexity constraint) suffer from the lack of traded options to calibrate the models, even for heavily traded markets such as the S&P 500. Moreover, the question of extrapolation is complex with such methods, as we are limited by the lowest and highest traded strikes. This is an issue, in particular for OTM calls, which are not as heavily traded as OTM puts.

We propose here another way to solve these issues. Defining an optimal network design, we insure that:

- our results respond on a daily basis to changes in the market conditions.
- the second derivative of the predictions is always positive.
- we can construct a volatility surface even if certain maturities are not traded, which enables us for example to construct time series of features of the RND for specific fixed maturities (e.g. 30 trading days).

More formally, let first define the price of a European call option with maturity $\tau = T - t$:

$$C = e^{r\tau} \int_{K}^{\infty} (S_T - K, 0)^{+} f(S_T) dS_T \tag{2}$$

Following Merton (1973) we place generic pricing upper and lower bounds:

$$S_t e^{-q\tau} \geqslant C \geqslant (S_t e^{-q\tau} - K e^{-r\tau}, 0)^{+} \tag{3}$$

where q is the continuously compounded dividend rate. Moreover, we impose constraints on strike and butterfly spread.

$$e^{-r\tau} \leqslant \frac{\partial C}{\partial K} \leqslant 0 \quad \text{and} \quad \frac{\partial^2 C}{\partial K^2} \geqslant 0 \tag{4}$$

We are particularly interested in the butterfly non-arbitrage condition. To understand it, let us consider a classical long butterfly strategy:

$$C_{K_1}^T - 2C_{K_2}^T + C_{K_3}^T > 0$$

in a no-arbitrage and competitive economy, if the strike prices are increasing $K_1 < K_2 < K_3$ and equidistant $K_2 = (K_1 + K_3)/2$ the price of this strategy needs to be strictly positive. Indeed, the payoff of the strategy is always non-negative (strictly positive when $S_T \in]K_1; K_3[$, null elsewhere). This butterfly condition is actually intimately linked with the convexity of option prices with respect to the strike. It means that both the second derivative of the price of calls and puts with respect of the strike must be positive From Equation (1) this second derivative expresses the RND, once adjusted for the time value of money. As this density is equal to the second derivative of the price of the option with respect to the strike price, we need this derivative to be positive. Therefore, to optimally design our neural network to fit the volatility surface and construct the RND, we need to take into account these features.

3 Basics of Neural Networks

One of the biggest explanation for the popularity of neural networks relates to the universal approximation theorem of Hornik et al. (1989). According to this theorem, under specific conditions regarding the activation function (notably being continuous and non-constant), a neural network is able to approximate continuous functions. Cybenko (1989) notably proved this result for the special case of the sigmoid activation function. However, this quality of approximation might need an explosive number parameters to deliver results. The constant increase in the performance of computing power and number of data were key elements to explain the impressive results achieved by neural networks in a large range of tasks, such as image classification.

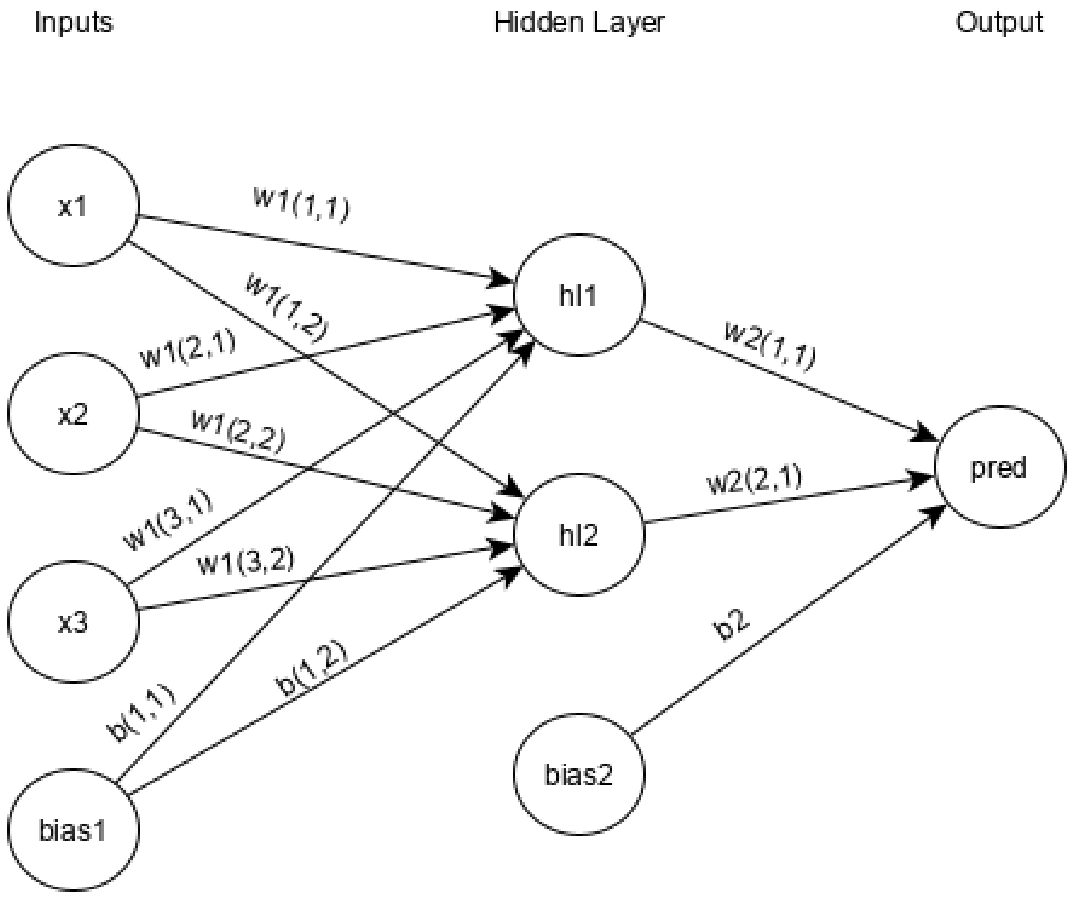

Figure 1: Example: Basic Neural Network with 1 Hidden Layer

An Artificial Neural Network (ANN) consists in one input layer, a flexible number of hidden layers (1 in the example of Figure 1, but can be null), and one output layer. The result of each node in the hidden layers/output layer is a simple linear combination of the previous layers, plus a bias, with an activation function. Taking the previous network, we can easily compute the value of each value in the hidden layer, and the final prediction:

$$hl_1 = \sigma_1\Big(\sum_{i=1}^{3} w^1_{i,1} x_i + b_{1,1}\Big)$$

$$hl_2 = \sigma_1\Big(\sum_{i=1}^{3} w_{i,2}^{1} x_i + b_{1,2}\Big)$$

$$pred = F(X) = \sigma_2\Big(\sum_{i=1}^{2} w_{i,1}^{2} hl_i + b_2\Big)$$

With σ_i the activation function applied to the i_th layer. The range of shapes for this activation function is very wide. It is extremely useful to introduce some nice properties in the neural network. First, because it can introduce some non-linearity. Second, because we can force the output of the network to be positive, or be convex with respect to one particular input, as in this paper.

The task of the model is to minimize some loss function, which is defined by the user. In this case, we choose to use a simple Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{1}{N}\sum_{i=1}^{N} \frac{|y_i - F(X_i)|}{y_i}$$

With $F(X_i)$ the result outputted by the neural network, and y_i the true value to be fitted. While this choice might be unorthodox, it is motivated by the fact that classical loss functions such as Mean Squared Error tend to overweight outliers. In addition, the fact that the gradient of the MAPE function only takes into account the sign of the error is not an issue as our neural network is very simple, with few parameters and a large amount of training data.

In the process of constructing the right model for a dedicated task, we should always have in mind the bias-variance dilemma. On one side, having too few parameters might prevent the model from learning interesting relationships between the input and the output, which would eventually lead to under-fitting. This is called the *bias*. On the other side, adding too many parameters in the model (which could be the case with deep learning techniques if regularization is not properly implemented) would lead to overfitting, meaning that the model would not be able to properly generalized to new data. This is called the *variance*.

The bias can be estimated using the value of the loss. To test the capacity to generalize to new data, we split our data in two categories: train and validation. As indicated by its name, the train set is used to optimize the weights of the model to minimize the loss. On the other side, the validation set is used to test the generalization capacity of the model, once all of the weights are fixed. A huge discrepancy in the loss between the train and validation sets is most likely to indicate overfitting. When working with classical machine learning tasks, such as classifying pictures, one can easily separate all the data between train and validation set, and can even shuffle all of the data and do the separation a several times. When working with time series, as it is most of the time the case in financial tasks, we can't reproduce the same methodology, as we can't feed the model with future data. We choose another strategy:

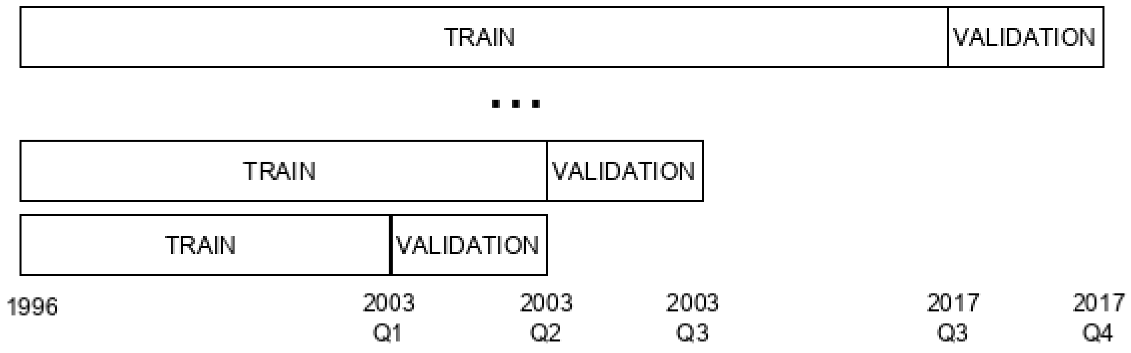

Figure 2: Train/Validation split with time series

Under this set-up, the model never sees any future data. A new model is retrained every quarter, with an expanding window. Then, the validation set is the next quarter.
Every quarter, the model is trained using back-propagation. The basic idea of this approach is to compute the gradient of the loss with respect to each individual weight, using the chain rule, one at a time. In particular, we use the RMSprop algorithm Tieleman and Hinton (2012) to avoid the vanishing gradient problem Hochreiter (1998), by dividing the gradient by the average of its recent magnitude.

4 Introducing the Convexity Constraint in the Neural Network

Dataset and training of the model We now propose our solution to the problem stated in Part 2. In the same spirit of Laurini (2011), we want to introduce five conditions on the fitted volatility surface. First, we want the volatility surface to not be static from one day to the next. This means introducing other input than the moneyness and the maturity. Second, we want to constraint the monotonicity of the implied volatility with respect to the moneyness.

Third, we want to introduce this convexity constraint, still with respect to the moneyness. Fourth, we do not want to introduce discontinuities by fitting some locations (such as deep OTM puts for example) of the surface. Finally, we want our method to be quick to run.
We define the moneyness m as the ratio between the strike K and the spot price of the underlying S_0:

$$m = \frac{K}{S_0} \tag{5}$$

The output of the neural network is the implied volatility of the option. The advantage in this database is that all of the implied volatilities are comprised between 0 and 1, which helps the gradient to converge. Second, using implied volatilities, we can compare all of the options of the database, no matter the price of the underlying S_0. We defined all the input of the network:

- m, the moneyness defined in (5)
- τ, the number of years before expiration (which in bounded between 0 and 1)
- V, the VIX index, which is normalized between 0 and 1
- rf, the risk-free rate, calculated using the LIBOR USD
- μ_1^{7d}, the average return during the last 7 days
- μ_2^{7d}, the standard deviation of the returns during the last 7 days
- μ_1^{30d}, the average return during the last 30 days
- μ_2^{30d}, the standard deviation of the returns during the last 30 days
- μ_3^{30d}, the skewness of the returns during the last 30 days
- μ_4^{30d}, the kurtosis of the returns during the last 30 days

We use the options from the OptionMetrics dataset, from 1999 to 2017. We apply some filters to the data, which are very similar in spirit to most of the studies of the RND, such as in Birru and Figlewski (2012). The idea behind these filters is straight-forward. First, even for the most liquid underlyings such as the S&P 500, many of the options do not trade heavily at all maturities and strikes. Thus, we need to be careful when using end-of-day prices. This low-volume activity is particularly true for in-the-money (ITM) options.

- minimum volume of 40 contracts exchanged on a given date
- minimum maturity of 7 days
- maximum maturity of 1 year
- minimum price of 0.05$
- Out-of-the-money (OTM) and at-the-money (ATM) options. For calls, we only keep options with a moneyness above 97%, and below 103% for puts

First model: shape and constraints We first present the basic model to include the convexity constraints into the neural network architeture. We stress the fact that this kind of models are easily generalizable to other input, as long as it does not depends on the moneyness. The basic model can be graphically summarized as follows:

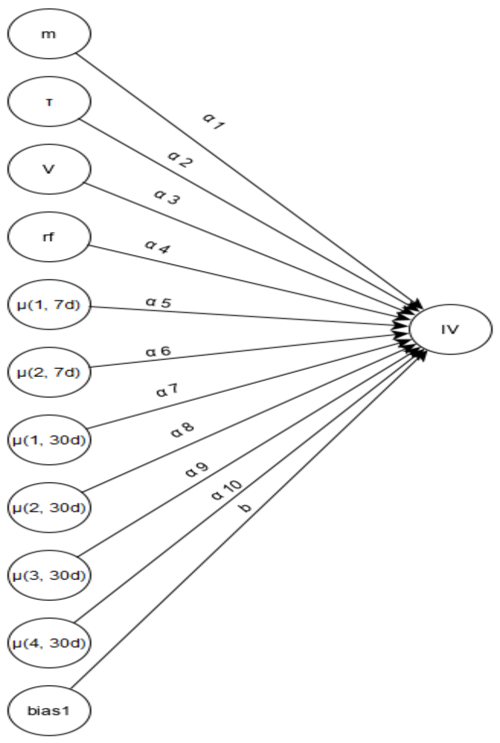

Figure 3: First model: no hidden layer

We note b as the bias. The weight applied to the i_{th} variable is denoted α_i. To compute the implied volatility (denoted ξ), we apply an activation function noted $\sigma(x)$. We note:

$$\xi = \sigma(b+\alpha_1 m+\alpha_2\tau+\alpha_3 V+\alpha_4 rf+\alpha_5\mu_1^{7d}+\alpha_6\mu_2^{7d}+\alpha_7\mu_1^{30d}+\alpha_8\mu_2^{30d}+\alpha_9\mu_3^{30d}+\alpha_{10}\mu_4^{30d}) \quad (6)$$

Introducing the notation f:

$$f = b+\alpha_1 m+\alpha_2\tau+\alpha_3 V+\alpha_4 rf+\alpha_5\mu_1^{7d}+\alpha_6\mu_2^{7d}+\alpha_7\mu_1^{30d}+\alpha_8\mu_2^{30d}+\alpha_9\mu_3^{30d}+\alpha_{10}\mu_4^{30d} \quad (7)$$

We can compute the first derivative of ξ with respect to the moneyness m:

$$\frac{\partial\xi}{\partial m} = \alpha_1\sigma'(f)$$

And the second derivative:

$$\frac{\partial^2\xi}{\partial m^2} = \alpha_1^2\sigma''(f) \quad (8)$$

So the necessary condition to get a convex implied volatility with respect to the moneyness m is that $\sigma''(f) > 0$. We now introduce the softplus activation function, which has one major advantage: the output of the function is always positive, as we want a positive IV.

$$\text{softplus} = \sigma(x) = log(1+e^x)$$

We compute the first derivative of the softplus activation function with respect to m:

$$\sigma(f) = log(1+e^{b+\alpha_1 m+\alpha_2\tau+\alpha_3 V+\alpha_4 rf+\alpha_5\mu_1^{7d}+\alpha_6\mu_2^{7d}+\alpha_7\mu_1^{30d}+\alpha_8\mu_2^{30d}+\alpha_9\mu_3^{30d}+\alpha_{10}\mu_4^{30d}}) \quad (9)$$

For the sake of clarity, we introduce a new variable θ:

$$\theta = b+\alpha_2\tau+\alpha_3 V+\alpha_4 rf+\alpha_5\mu_1^{7d}+\alpha_6\mu_2^{7d}+\alpha_7\mu_1^{30d}+\alpha_8\mu_2^{30d}+\alpha_9\mu_3^{30d}+\alpha_{10}\mu_4^{30d}$$

(9) now becomes:

$$\sigma(f) = log(1+e^{\theta+\alpha_1 m}) \quad (10)$$

The first derivative becomes:

$$\sigma'(f) = \alpha_1 \frac{e^{\theta+\alpha_1 m}}{1+e^{\theta+\alpha_1 m}}$$

We now compute the second derivative:

$$\sigma''(f) = \alpha_1 \frac{\alpha_1 e^{\theta+\alpha_1 m}(1+e^{\theta+\alpha_1 m}) - e^{\theta+\alpha_1 m}\alpha_1 e^{\theta+\alpha_1 m}}{(1+e^{\theta+\alpha_1 m})^2}$$

$$\Leftrightarrow \sigma''(f) = \frac{\alpha_1^2 e^{\theta+\alpha_1 m}}{(1+e^{\theta+\alpha_1 m})^2} > 0 \tag{11}$$

For this first model, the use of the softplus activation function ensures an IV ξ convex with respect to the moneyness.

Second model: shape and constraints The first model is extremely simple and has some interesting properties. However it suffers from two drawbacks, which are linked. First, with no hidden layer, the predictive performance of the model might be poor (even if it could be increased by the use of other inputs). Second, using a softplus activation function with no hidden layer is actually quite similar to implementing a logistic regression. We now propose another model, this time with the use of a hidden layer.

We present the results in the case of two nodes in the hidden layer. However, these results are easily generalizable to any number of nodes. The notations stay the same, with the same initial inputs and σ the first activation function. However, we now introduce a second activation function ϕ, which maps the hidden layer to the final output ξ. The bias at the input level is denoted b_0 and the one at the hidden layer level b_1. The neural network becomes:

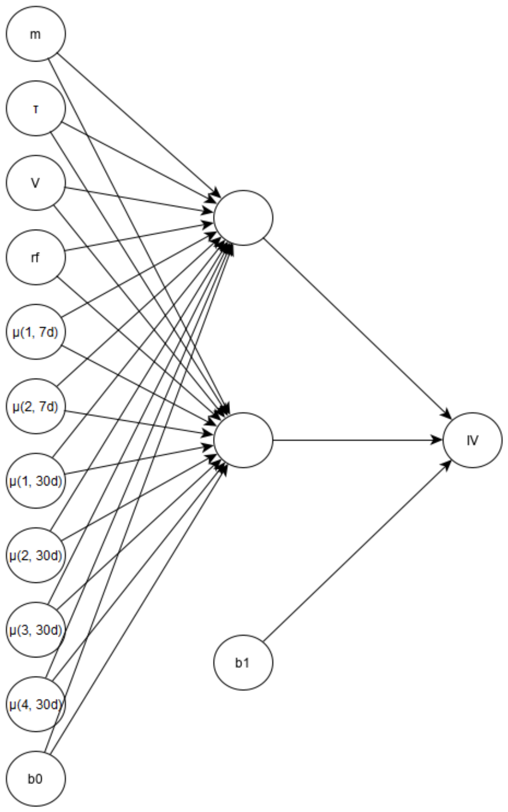

Figure 4: Second model: one hidden layer

The weight applied to the i_{th} feature for the N_{th} node in the hidden layer is denoted α_{Ni}. The weight applied to the nodes in the hidden layer to predict the standardized price ξ are denoted γ_N.

General case We introduce the output of the neural network with one hidden layer, ξ:

$$xi = \phi(b_1 + \gamma_1\sigma(\theta_1 + \alpha_{1,1}m) + \gamma_2\sigma(\theta_2 + \alpha_{2,1}m)) \tag{12}$$

We take the first derivative of ξ with respect to the moneyness m:

$$\frac{\partial \xi}{\partial m} = \left[\gamma_1\alpha_{1,1}\sigma'(\theta_1 + \alpha_{1,1}m) + \gamma_2\alpha_{2,1}\sigma'(\theta_2 + \alpha_{2,1}m)\right]\phi'(b_1 + \gamma_1\sigma(\theta_1 + \alpha_{1,1}m) + \gamma_2\sigma(\theta_2 + \alpha_{2,1}m)) \tag{13}$$

And the second derivative:

$$\frac{\partial^2 \xi}{\partial m^2} = \Big[\gamma_1 \alpha_{1,1}^2 \sigma''(\theta_1 + \alpha_{1,1} m) + \gamma_2 \alpha_{2,1}^2 \sigma''(\theta_2 + \alpha_{2,1} m)\Big] \phi'\Big(b_1 + \gamma_1 \sigma(\theta_1 + \alpha_{1,1} m) + \gamma_2 \sigma(\theta_2 + \alpha_{2,1} m)\Big) + \Big[\gamma_1 \alpha_{1,1} \sigma'(\theta_1 + \alpha_{1,1} m) + \gamma_2 \alpha_{2,1} \sigma'(\theta_2 + \alpha_{2,1} m)\Big]^2 \phi''\Big(b_1 + \gamma_1 \sigma(\theta_1 + \alpha_{1,1} m) + \gamma_2 \sigma(\theta_2 + \alpha_{2,1} m)\Big)$$

Softplus activation function We now suppose that both ϕ and σ are softplus activation functions. We have:

$$\phi\Big(b_1 + \gamma_1 \sigma(\theta_1 + \alpha_{1,1} m) + \gamma_2 \sigma(\theta_2 + \alpha_{2,1} m)\Big) = log(1 + e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})})$$

Its first derivative is:

$$\frac{\partial \phi}{\partial m} = \frac{\Big(\frac{\gamma_1 \alpha_{1,1} e^{\theta_1 + \alpha_{1,1} m}}{1+e^{\theta_1 + \alpha_{1,1} m}} + \frac{\gamma_2 \alpha_{2,1} e^{\theta_2 + \alpha_{2,1} m}}{1+e^{\theta_2 + \alpha_{2,1} m}}\Big) e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})}}{1 + e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})}}$$

The sign of the derivative is only determined by $\gamma_1 \alpha_{1,1}$ and $\gamma_2 \alpha_{2,1}$. Chollet et al. (2015) show that by constraining the weights, we can make sure that $\frac{\partial \phi}{\partial m} < 0$ for calls, and $\frac{\partial \phi}{\partial m} > 0$ for puts.

Now the second derivative:

$$\frac{\partial^2 \phi}{\partial m^2} = e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})} \frac{\Big(-\frac{\gamma_1 \alpha_{1,1}^2 e^{2(\theta_1 + \alpha_{1,1} m)}}{(1+e^{\theta_1 + \alpha_{1,1} m})^2} + \frac{\gamma_1 \alpha_{1,1}^2 e^{\theta_1 + \alpha_{1,1} m}}{1+e^{\theta_1 + \alpha_{1,1} m}} - \frac{\gamma_2 \alpha_{2,1}^2 e^{2(\theta_2 + \alpha_{2,1} m)}}{(1+e^{\theta_2 + \alpha_{2,1} m})^2} + \frac{\gamma_2 \alpha_{2,1}^2 e^{\theta_2 + \alpha_{2,1} m}}{1+e^{\theta_2 + \alpha_{2,1} m}}\Big)}{1 + e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})}} + \frac{\Big(\frac{\gamma_1 \alpha_{1,1} e^{\theta_1 + \alpha_{1,1} m}}{1+e^{\theta_1 + \alpha_{1,1} m}} + \frac{\gamma_2 \alpha_{2,1} e^{\theta_2 + \alpha_{2,1} m}}{1+e^{\theta_2 + \alpha_{2,1} m}}\Big)^2 e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})}}{1 + e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})}} - \frac{\Big(\frac{\gamma_1 \alpha_{1,1} e^{\theta_1 + \alpha_{1,1} m}}{1+e^{\theta_1 + \alpha_{1,1} m}} + \frac{\gamma_2 \alpha_{2,1} e^{\theta_2 + \alpha_{2,1} m}}{1+e^{\theta_2 + \alpha_{2,1} m}}\Big)^2 e^{2(b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m}))}}{\Big(1 + e^{b_1 + \gamma_1 log(1+e^{\theta_1 + \alpha_{1,1} m}) + \gamma_2 log(1+e^{\theta_2 + \alpha_{2,1} m})}\Big)^2}$$

We need $\frac{\partial^2 \phi}{\partial m^2} \geqslant 0$. Let's decompose this expression (assuming that both γ_1 and γ_2 are set to be positive):

$$\Big(-\frac{\gamma_1\alpha_{1,1}^2 e^{2(\theta_1+\alpha_{1,1}m)}}{(1+e^{\theta_1+\alpha_{1,1}m})^2}+\frac{\gamma_1\alpha_{1,1}^2 e^{\theta_1+\alpha_{1,1}m}}{1+e^{\theta_1+\alpha_{1,1}m}}-\frac{\gamma_2\alpha_{2,1}^2 e^{2(\theta_2+\alpha_{2,1}m)}}{(1+e^{\theta_2+\alpha_{2,1}m})^2}+\frac{\gamma_2\alpha_{2,1}^2 e^{\theta_2+\alpha_{2,1}m}}{1+e^{\theta_2+\alpha_{2,1}m}}\Big)$$

$$\Leftrightarrow \gamma_1\alpha_{1,1}^2\Big(\frac{e^{\theta_1+\alpha_{1,1}m}}{1+e^{\theta_1+\alpha_{1,1}m}}-\frac{e^{\theta_1+\alpha_{1,1}m}}{(1+e^{\theta_1+\alpha_{1,1}m})^2}\Big)+\gamma_2\alpha_{2,1}^2\Big(\frac{e^{\theta_2+\alpha_{2,1}m}}{1+e^{\theta_2+\alpha_{2,1}m}}-\frac{e^{\theta_2+\alpha_{2,1}m}}{(1+e^{\theta_2+\alpha_{2,1}m})^2}\Big) \quad (14)$$

As $1+e^{\theta_1+\alpha_{1,1}m}>1$, $1+e^{\theta_1+\alpha_{1,1}m}<(1+e^{\theta_2+\alpha_{2,1}m})^2$. Thus, (14) is always positive.

Now focusing on the other side of the expression:

$$\frac{\Big(\frac{\gamma_1\alpha_{1,1}e^{\theta_1+\alpha_{1,1}m}}{1+e^{\theta_1+\alpha_{1,1}m}}+\frac{\gamma_2\alpha_{2,1}e^{\theta_2+\alpha_{2,1}m}}{1+e^{\theta_2+\alpha_{2,1}m}}\Big)^2 e^{b_1+\gamma_1 log(1+e^{\theta_1+\alpha_{1,1}m})+\gamma_2 log(1+e^{\theta_2+\alpha_{2,1}m})}}{1+e^{b_1+\gamma_1 log(1+e^{\theta_1+\alpha_{1,1}m})+\gamma_2 log(1+e^{\theta_2+\alpha_{2,1}m})}}$$
$$-\frac{\Big(\frac{\gamma_1\alpha_{1,1}e^{\theta_1+\alpha_{1,1}m}}{1+e^{\theta_1+\alpha_{1,1}m}}+\frac{\gamma_2\alpha_{2,1}e^{\theta_2+\alpha_{2,1}m}}{1+e^{\theta_2+\alpha_{2,1}m}}\Big)^2 e^{2(b_1+\gamma_1 log(1+e^{\theta_1+\alpha_{1,1}m})+\gamma_2 log(1+e^{\theta_2+\alpha_{2,1}m}))}}{\Big(1+e^{b_1+\gamma_1 log(1+e^{\theta_1+\alpha_{1,1}m})+\gamma_2 log(1+e^{\theta_2+\alpha_{2,1}m})}\Big)^2}$$

First noting x such as:

$$x=b_1+\gamma_1 log(1+e^{\theta_1+\alpha_{1,1}m})+\gamma_2 log(1+e^{\theta_2+\alpha_{2,1}m})$$

Factorizing, the expression becomes:

$$\frac{\Big(\frac{\gamma_1\alpha_{1,1}e^{\theta_1+\alpha_{1,1}m}}{1+e^{\theta_1+\alpha_{1,1}m}}+\frac{\gamma_2\alpha_{2,1}e^{\theta_2+\alpha_{2,1}m}}{1+e^{\theta_2+\alpha_{2,1}m}}\Big)^2}{1+e^x}\Big(e^x-\frac{e^{2x}}{1+e^x}\Big)$$

Which is strictly positive. In conclusion, assuming both γ_1 and γ_2 are positive, the second derivative is strictly positive.

5 Practical Implementation: Volatility Surface

Now that we have established how to introduce the convexity in the output of the neural network with respect to the level of moneyness, to get a free of butterfly arbitrage volatility surface, we present the results of the model. We start by establishing some descriptive statistics:

Table 1: Descriptive Statistics

YEAR	CALLS	PUTS
1999	8 855	13 984
2000	8 698	10 843
2001	8 532	11 334
2002	8 963	12 806
2003	10 236	14 491
2004	11 147	17 274
2005	12 406	17 922
2006	14 144	21 344
2007	16 648	26 117
2008	23 236	30 489
2009	22 061	31 153
2010	23 215	36 388
2011	25 730	40 127
2012	30 350	47 480
2013	39 246	61 589
2014	44 799	82 903
2015	56 700	104 529
2016	73 169	127 905
2017	84 539	164 588
Total	523 604	873 266
Average IV	13.92%	24.37%
St.Dev. IV	7.49%	21.40%

The first thing to notice is that the number of options heavily increases with the years. Thus, when computing the final loss statistics (MSE, MAE or MAPE), as we train a new model every quarter, we will weight the results by the number of options, to put more weight on periods with more options. We start with a set of hyperparameters we tested. As the procedure is relatively quick, we were able to test every possible combination (= 384) for the hyperparameters.

Table 2: List of Possible Hyperparameters

Hyperparameters	Options
Number of neurons in the hidden layer	[3 ; 4; 5; 6]
Initialization	[normal ; uniform ; lecun_normal ; lecun_uniform]
Optimizer	[SGD ; RMSprop]
Number of epochs	[30 ; 40; 50 ; 60]
Batch Size	[128 ; 256 ; 512]

For each train/validation set pair, we compute three common loss measures, the Mean-Absolute-Error (MAE), the Mean-Absolute-Percentage-Error (MAPE) and the Mean-Squared-Error (MSE), which are defined as:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - F(X_i)|}{y_i}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\Big|y_i - F(X_i)\Big|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\Big(y_i - F(X_i)\Big)^2$$

We then weight each loss value by the number of options of the period, to compute two final statistics (MAE and MAPE) per model. Obviously, we choose the model with the lowest loss value, checking that there is no overfitting. The hyperparameters of the best model are found to be:

Table 3: List of Hyperparameters

Hyperparameters	Options
Number of neurons in the hidden layer	3
Initialization	lecun_normal
Optimizer	RMSprop
Number of epochs	50
Batch Size	128

As said above, the computation time is quite quick. For a basic configuration, such as a Intel(R) Core(TM) i3-7100 CPU with a RAM memory of 16Go, the whole process (one neural network per quarter, from 1999 to 2017, with one hidden layer) takes approximately 55 minutes for calls and 85 minutes for puts. Under the same setup, training the longer model (training period = 1999 Q1 to 2017 Q3) takes approximately 4 minutes for calls and 6.5 minutes for puts.

The results of our approach are difficult to benchmark for several reasons. First, we are able to study any maturity/level of moneyness, at any date, whereas cubic splines with a convexity constraint are quite limited by the number of options that are actively traded. This limitation is common to other techniques, such as SVI. We compare our approach anyway to the standard benchmark defined in **?**, where the implied volatility is modeled using quadratic polynomials (m being the moneyness and τ the maturity):

$$IV = \beta_0 + \beta_1 m + \beta_2 m^2 + \beta_1 \tau + \beta_1 \tau^2 + \beta_1 m\tau$$

In addition, to the best of our knowledge, our approach is the only one to use a neural network and add theoretical insurance that the resulted implied volatility will be convex with respect to the level of moneyness. This approach does not allow us to add a infinite number of hidden layers, or to use some common activation functions such as ReLU, which tend to have better results in terms of minimization of the loss, but do not give satisfaction in terms of conditions of smoothness for the obtained volatility surface.

Table 4: Loss: All of the data - Puts (*hl* = *hidden layer*)

LOSS	TRAIN	TRAIN	VALIDATION	VALIDATION
HL	no	yes	no	yes
MSE	4.68 x 10^{-03}	1.83 x 10^{-03}	5.20 x 10^{-03}	1.77 x 10^{-03}
MAE	3.51 x 10^{-02}	1.90 x 10^{-02}	3.84 x 10^{-02}	2.08 x 10^{-02}
MAPE	1.22 x 10^{-01}	6.72 x 10^{-02}	1.49 x 10^{-01}	7.74 x 10^{-02}

Table 5: Loss: All of the data - Calls (*hl* = *hidden layer*)

LOSS	TRAIN	TRAIN	VALIDATION	VALIDATION
HL	no	yes	no	yes
MSE	2.34×10^{-03}	1.46×10^{-03}	4.51×10^{-03}	1.04×10^{-03}
MAE	2.13×10^{-02}	1.54×10^{-02}	2.23×10^{-02}	1.51×10^{-02}
MAPE	1.13×10^{-01}	8.73×10^{-02}	1.45×10^{-01}	1.10×10^{-01}

Table 6: Loss: All of the data - Benchmark (Quadratic Polynomials)

LOSS	TRAIN	TRAIN	VALIDATION	VALIDATION
Type	calls	puts	calls	puts
MSE	3.97×10^{-03}	4.86×10^{-03}	3.87×10^{-03}	4.73×10^{-03}
MAE	4.70×10^{-02}	5.17×10^{-02}	4.95×10^{-02}	5.03×10^{-02}
MAPE	3.19×10^{-01}	2.17×10^{-01}	4.52×10^{-01}	2.65×10^{-01}

The first thing to notice is that globally, the performance of the validation set is a little less good than the train set, which is normal. However, the discrepancy is quite small, meaning that our model seems to be quite good at generalizing to new data. So, we avoided overfitting. Second, both for calls and puts, adding a hidden layer seems to significantly increase the performance, which is also normal, as adding parameters decrease the bias. Third, it seems that the model, after the fine tuning of hyperparameters, performs better for puts than for calls. We then perform an analysis to get a clearer idea of which kind of options weight negatively on the global loss in the models. Finally, we tend to largely over-perform the standard quadratic polynomials benchmark, for both calls and puts.
We define three kinds of maturities: short-term (maturity $\leqslant$ 30 days), mid-term (30 days $<$ maturity $\leqslant$ 180 days), and long-term (maturity $>$ 180 days). We also consider two kinds of level of moneyness: at-the-money (ATM, moneyness > 0.9 for puts, moneyness < 1.1 for calls), and out-of-the-money (OTM, moneyness $\leqslant 0.9$ for puts, moneyness $\geqslant 1.1$ for calls). We emphasise that the definition of ATM/OTM options is quite large, but as we only work with OTM options (as defined previously), we want a strong distinction between the levels of moneyness. We do not need to present the results of the train set, as it is exactly the same neural network as presented in the last two tables.

Table 7: Loss: Long-term & ATM ($hl = hidden\ layer,\ bench = benchmark$)

LOSS	PUTS	PUTS	PUTS (bench)	CALLS	CALLS	CALLS (bench)
HL	no	yes	/	no	yes	/
MSE	$3.07 \text{ x } 10^{-03}$	$1.21 \text{ x } 10^{-03}$	$3.80 \text{ x } 10^{-03}$	$4.17 \text{ x } 10^{-03}$	$3.73 \text{ x } 10^{-04}$	$2.81 \text{ x } 10^{-03}$
MAE	$2.98 \text{ x } 10^{-02}$	$2.02 \text{ x } 10^{-02}$	$4.92 \text{ x } 10^{-02}$	$1.79 \text{ x } 10^{-02}$	$1.14 \text{ x } 10^{-02}$	$4.46 \text{ x } 10^{-02}$
MAPE	$1.29 \text{ x } 10^{-01}$	$8.85 \text{ x } 10^{-02}$	$2.45 \text{ x } 10^{-01}$	$9.32 \text{ x } 10^{-02}$	$7.68 \text{ x } 10^{-02}$	$3.42 \text{ x } 10^{-01}$

Table 8: Loss: Long-term & OTM (hl = *hidden layer*, $bench$ = *benchmark*)

LOSS	PUTS	PUTS	PUTS (bench)	CALLS	CALLS	CALLS (bench)
HL	no	yes	/	no	yes	/
MSE	$1.06 \text{ x } 10^{-02}$	$1.41 \text{ x } 10^{-03}$	$3.79 \text{ x } 10^{-03}$	$1.04 \text{ x } 10^{-02}$	$6.78 \text{ x } 10^{-03}$	$3.54 \text{ x } 10^{-03}$
MAE	$4.95 \text{ x } 10^{-02}$	$2.33 \text{ x } 10^{-02}$	$4.38 \text{ x } 10^{-02}$	$3.28 \text{ x } 10^{-02}$	$1.70 \text{ x } 10^{-02}$	$4.70 \text{ x } 10^{-02}$
MAPE	$1.32 \text{ x } 10^{-01}$	$7.06 \text{ x } 10^{-02}$	$1.45 \text{ x } 10^{-01}$	$1.58 \text{ x } 10^{-01}$	$1.11 \text{ x } 10^{-01}$	$3.47 \text{ x } 10^{-01}$

Table 9: Loss: Mid-term & ATM (hl = *hidden layer*, $bench$ = *benchmark*)

LOSS	PUTS	PUTS	PUTS (bench)	CALLS	CALLS	CALLS (bench)
HL	no	yes	/	no	yes	/
MSE	$2.12 \text{ x } 10^{-03}$	$1.10 \text{ x } 10^{-03}$	$3.60 \text{ x } 10^{-03}$	$3.00 \text{ x } 10^{-03}$	$5.01 \text{ x } 10^{-04}$	$3.72 \text{ x } 10^{-03}$
MAE	$1.86 \text{ x } 10^{-02}$	$1.24 \text{ x } 10^{-02}$	$4.69 \text{ x } 10^{-02}$	$1.69 \text{ x } 10^{-02}$	$1.11 \text{ x } 10^{-02}$	$4.99 \text{ x } 10^{-02}$
MAPE	$9.98 \text{ x } 10^{-02}$	$5.88 \text{ x } 10^{-02}$	$2.78 \text{ x } 10^{-01}$	$1.16 \text{ x } 10^{-01}$	$8.76 \text{ x } 10^{-02}$	$4.65 \text{ x } 10^{-01}$

Table 10: Loss: Mid-term & OTM (hl = *hidden layer*, $bench$ = *benchmark*)

LOSS	PUTS	PUTS	PUTS (bench)	CALLS	CALLS	CALLS (bench)
HL	no	yes	/	no	yes	/
MSE	$5.99 \text{ x } 10^{-03}$	$1.78 \text{ x } 10^{-03}$	$5.59 \text{ x } 10^{-03}$	$2.18 \text{ x } 10^{-02}$	$2.75 \text{ x } 10^{-03}$	$5.55 \text{ x } 10^{-03}$
MAE	$3.82 \text{ x } 10^{-02}$	$2.25 \text{ x } 10^{-02}$	$6.11 \text{ x } 10^{-02}$	$6.28 \text{ x } 10^{-02}$	$3.08 \text{ x } 10^{-02}$	$5.61 \text{ x } 10^{-02}$
MAPE	$1.00 \text{ x } 10^{-01}$	$5.94 \text{ x } 10^{-02}$	$1.97 \text{ x } 10^{-01}$	$2.31 \text{ x } 10^{-01}$	$1.55 \text{ x } 10^{-01}$	$3.33 \text{ x } 10^{-01}$

Table 11: Loss: Short-term & ATM (hl = *hidden layer*, $bench$ = *benchmark*)

LOSS	PUTS	PUTS	PUTS (bench)	CALLS	CALLS	CALLS (bench)
HL	no	yes	/	no	yes	/
MSE	1.96 x 10^{-03}	1.11 x 10^{-03}	3.98 x 10^{-03}	2.38 x 10^{-03}	1.11 x 10^{-03}	3.43 x 10^{-03}
MAE	2.82 x 10^{-02}	1.63 x 10^{-02}	5.29 x 10^{-02}	2.00 x 10^{-02}	1.58 x 10^{-02}	4.77 x 10^{-02}
MAPE	1.82 x 10^{-01}	9.00 x 10^{-02}	3.64 x 10^{-01}	1.66 x 10^{-01}	1.28 x 10^{-01}	4.84 x 10^{-01}

Table 12: Loss: Short-term & OTM ($hl = hidden\ layer,\ \ bench = benchmark$)

LOSS	PUTS	PUTS	PUTS (bench)	CALLS	CALLS	CALLS (bench)
HL	no	yes	/	no	yes	/
MSE	1.90 x 10^{-02}	5.52 x 10^{-03}	8.40 x 10^{-03}	3.88 x 10^{-02}	1.13 x 10^{-02}	1.80 x 10^{-02}
MAE	1.13 x 10^{-01}	4.99 x 10^{-02}	5.88 x 10^{-02}	1.01 x 10^{-01}	6.15 x 10^{-02}	8.40 x 10^{-02}
MAPE	2.77 x 10^{-01}	1.17 x 10^{-01}	1.36 x 10^{-01}	2.88 x 10^{-01}	1.91 x 10^{-01}	2.36 x 10^{-01}

We find similar patterns for both Calls and Puts. Once again, our approach tend to largely over-perform the traditional quadratic polynomial benchmark. Globally, we find that the results are less good for OTM options compared to ATM. In the meantime, the longer the maturity, the better the results. These results are not surprising and can be easily explained. Indeed, if we look at the standard deviation of the implied volatility for the different locations in the volatility surface, we find an enormous discrepancies. For example, in the case of Calls, the standard deviation of OTM options is 30% higher compared to ATM options in the case of mid-term, and 64% higher for long-term options. In the same fashion, the implied volatility is 4.3 time higher for short-term options compared with long-term ones.

Aggregating losses at different time horizons, all results presented so far are unconditional. While unconditional results are good to provide a broad overview of the analysis, we now focus on the time series to identify periods of higher/lower accuracy. Figure 5 depicts the time series of realized and predicted average IV extracted from European Call options. Divided vertically, the left (right) panel presents the predicted IV estimated without (with) the hidden layer. The same holds for Figure 6, but for European Put options.

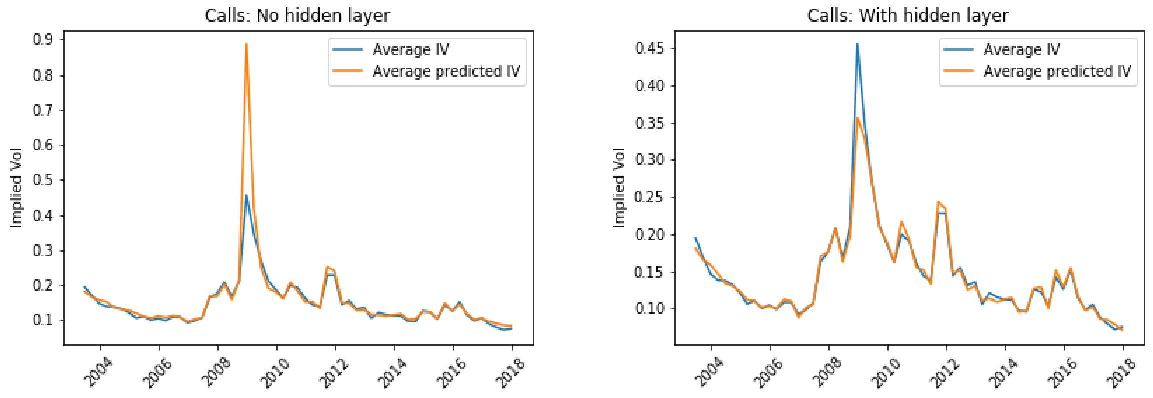

Figure 5: **Time series** of realized (in blue) and predicted (in orange) average IV from Call options. The left (right) panels depicts the predicted average IV estimated without (with) hidden layer.

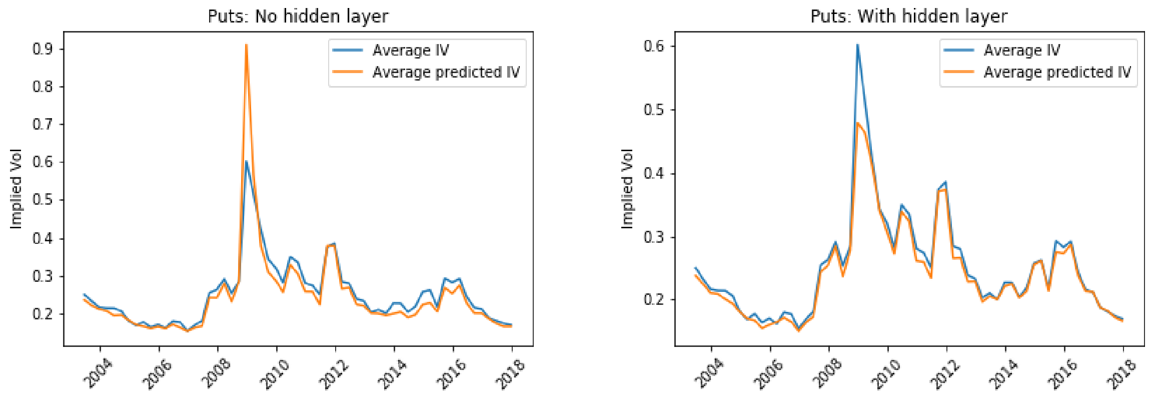

Figure 6: **Time series** of realized (in blue) and predicted (in orange) average IV from Put options. The left (right) panels depicts the predicted average IV estimated without (with) hidden layer.

Once again, the pattern are extremely similar between Calls and Puts. We have the confirmation that adding a hidden layer is necessary to increase the quality of the fit. We also see that the results for the 2008 period are not good. This is quite understandable, as this kind of movements in the volatility surface were outstandingly new. What is reassuring is how the different new spikes in the volatility in the past few years (2011, 2015 and 2016) were properly fitted.

6 Practical Implementation: RND Estimation

Now that we established how to fit the volatility surface, we can focus on the estimation of the RND. Each quarter, a new neural network is trained and its weights are fixed. We are able to fit a complete volatility surface by changing two of the inputs of the network (the maturity and the level of moneyness). All other inputs (notably the VIX and the different moments of the past returns) are common to all of the options, and are variable from one day to the next. We are able to focus on any maturity on the volatility surface. We choose to focus in particular on the 30-day maturity. Once all of the implied volatility are fitted (still for a fixed maturity), we can deduce the price of the corresponding options using Black and Scholes (1973) to get a continuum in the moneyness-price space. Finally, as we use two separate neural networks (for calls and puts), we might have an issue of discontinuities for a level of moneyness of 1. To tackle this issue, we take advantage of the put-call parity:

$$C(t,T,K) - P(t,T,K) = S_t - Ke^{-rT} \tag{15}$$

where $C(t,T,K)$ and $P(t,T,K)$ are the price of a European Call and Put option at time t, for a maturity of T and with a strike price of K, respectiveli, S_t is the current price of the underlying and r the risk-free rate.

Thus, we are able to fit the complete volatility surface for both Puts and Calls, and translate the price of Calls to Puts, and the ones of Puts to Calls through Equation (15). We then use a simple rule to derive the final RND. For levels of moneyness equal or below 1, we simply take the average of the price of Puts derived from the dedicated neural network, and the price of Puts derived from Calls using (15). For levels of moneyness above 1, we take the average of the price of calls derived from the dedicated neural network, and the price of Calls derived from Puts using (15). Thanks to this method, we have a smoothness in the second derivative. We finally use the equation of the RND as defined in the Appendix. Also, we only focus on the neural networks using a hidden layer, as it tends to have far better results.

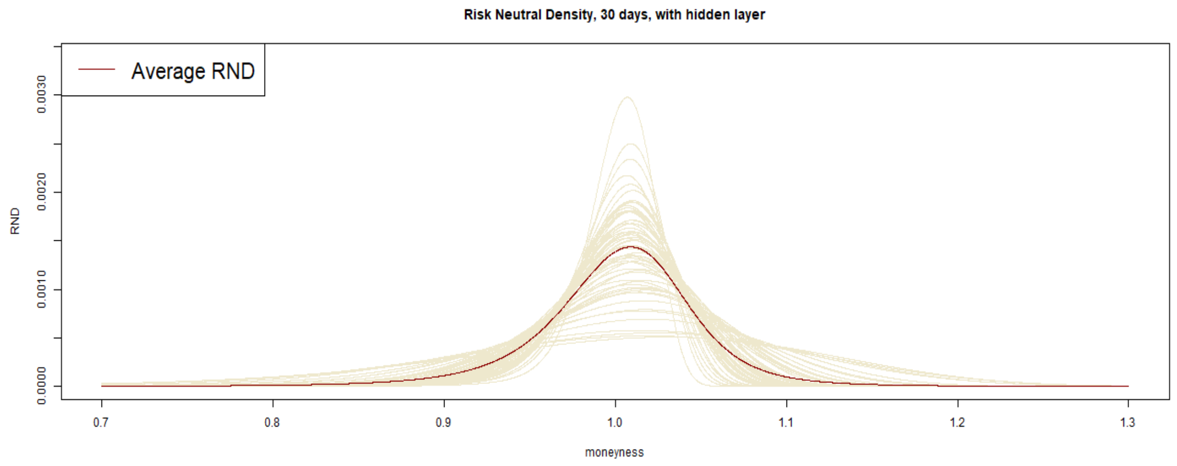

Figure 7: Average RND (maturity = 30 days)

We present here the evolution of the RND from 2003 to 2017. The idea is also to show that we are able to fit it everyday, without any discontinuity. Unfortunately, we had to only display one density per month (in orange): plotting 3716 densities in the same plot tend to be murky. We also plot the average RND over the whole period (in red). We see in Figure 7 that depending on the date, we can find very different shapes for the Risk-Neutral Densities. In particular, we find that the kurtosis moves over time.

7 Conclusion

When ones tries to construct robust estimations of the RND for a constant volatility, the main problem comes from the incapacity of typical methods to deal with the lack of data. For example, the use of cubic splines with a convexity constraint is a nice way to fit all of the points, but the different techniques of extrapolation (for example the use of clamped splines) fail to deliver proper results. With the large increase of popularity in machine learning techniques such as neural networks, we saw a new trend in research to get non-parametric estimations of the volatility surface. Unfortunately, the main interest has been on increasing as much as possible the accuracy of the fit, neglecting other parameters such as the non-arbitrage conditions. In particular, computing the RND requires a strict convexity of the price of the options with respect to the moneyness. Our approach tackles this issue. Moreover, using the put-call parity, we are able to avoid any discontinuity between calls and puts for a level of moneyness of 1.

8 Bibliography

Äit-Sahalia, Y. and J. Duarte (2003). Non-parametric option pricing under shape restrictions. *Journal of Econometrics 116*, 9–47.

Äit-Sahalia, Y. and A. W. Lo (1998). Non-parametric estimation of state-price densities implicit in financial asset prices. *The Journal of Finance 53*(2), 499–457.

Andreou, P. C., C. Charalambous, and S. H. Martzoukos (2008). Pricing and trading european options by combining artificial neural networks and parametric models with implied parameters. *European Journal of Operational Research 185*, 1415–1433.

Bates, D. S. (1996). Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *The Review of Financial Studies 9*(1), 69–107.

Bates, D. S. (Jul 1991). The crash of '87: was it expected? the evidence from options markets. *The Journal of Finance 46*, 1009–1044.

Birru, J. and S. Figlewski (May 2012). Anatomy of a meltdown: the risk neutral density for the s&p500 in the fall of 2008. *Journal of Financial Markets 15*, 151–180.

Black, F. and M. Scholes (1973). The pricing of options and corporate liabilities. *Journal of Political Economy 81*, 637–654.

Bliss, R. and N. Panigirtzoglou (2004). Option-implied risk aversion estimates. *Journal of Finance 59*, 407–446.

Bondarenko, O. (2003). Estimation of risk-neutral densities using positive convolution approximation. *Journal of Econometrics 116*(1), 85 – 112. Frontiers of financial econometrics and financial engineering.

Breeden, D. T. and R. H. Litzenberger (1978). Prices of state-contingent claims implicit in option prices. *The Journal of Business 51*, 621–651.

Buchen, P. and M. Kelly (1996). The maximum entropy distribution of an asset inferred from option prices. *Journal of Finance 31*, 143–159.

Campa, J. M., K. Chang, and R. Reider (1998). Implied exchange rate distributions: Evidence from otc option markets. *Journal of International Money and Finance 17*, 117–160.

Chollet, F. et al. (2015). Keras.

Christian, H. (2008). Implied volatility surface: construction methodologies and characteristics. *185*, 1415–1433.

Christoffersen, P., K. Jacobs, and B. Y. Chang (2013). Forecasting with option-implied information. *Handbook of Economic Forecasting 2*, 581–656.

Crespo, P. (2018). A random forest model averaging solution for implied volatility interpolation under small data sets.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems 2*, 303–314.

Derman, E. and I. Kani (1994). The volatility smile and its implied tree. *Quantitative Strategies Research Notes (Goldman Sachs, New York).*

Dupire, B. (1994). Pricing with a smile. *Risk 7*, 18–20.

Fengler, M. R. (2011, Jul). Option data and modeling bsm implied volatility. *Handbook of Computational Finance*, 117–142.

Giacomini, R., A. Gottschling, C. Haefke, and H. White (2008). Mixtures of t-distributions for finance and forecasting. *Journal of Econometrics 144*, 175–192.

Heston, S. L. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies 6*(2), 327–343.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6*, 107–116.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 359–366.

Hull, J. and A. White (1987). The pricing of options on assets with stochastic volatilities. *Journal of Finance 42*(2), 281–300.

Jackwerth, J. (1999). Option-implied risk-neutral distributions and implied binomial trees. *The Journal of Derivatives 7*(2), 66–82.

Jackwerth, J. (2000). Recovering risk aversion from option prices and realized returns. *The Review of Financial Studies 13*, 433–451.

Jackwerth, J. and M. Rubinstein (1996). Recovering probability distributions from option prices. *Journal of Finance 51*(5), 1611–1631.

Jarrow, R. and A. Rudd (1982). Approximate option valuation for arbitrary stochastic processes. *Journal of Financial Economics 10*(3), 347 – 369.

Laurini, M. P. (2011). Imposing no-arbitrage conditions in implied volatilities using constrained smoothing splines. *Applied Stochastic Models in Business and Industry 27*, 649–659.

Longstaff, F. (2015). Option pricing and the martingale restriction. *The Review of Financial Studies 8*(4), 1091–1124.

Ludwig, M. (2015). Robust estimation of shape-constrained state price density surfaces. *The Journal of Derivatives 22*(3), 56–72.

Madan, D. B. and F. Milne (1994). Contingent claims valued and hedged by pricing and investing in a basis. *Mathematical Finance 4*(3), 223–245.

Malliaris, M. and L. Salchenberger (1993). A neutal network model for estimating option prices. *Applied Intelligence 3*, 193–206.

Malz, A. (1997). Estimating the probability distribution of the future exchange rate from option prices. *The Journal of Derivatives 5*(2), 18–36.

Mostafa, F., T. Dillon, and E. Chang (2015). Computational intelligence approach to capturing the implied volatility. *Proc. IFIP International Conference on Artificial Intelligence in Theory and Practice*, 85–97.

Pironneau, O. (2019). Calibration of heston model with keras.

Rebonato, R. (2013). *Volatility and Correlation: The Perfect Hedger and the Fox.* While Online Library.

Ritchey, R. (1990). Call option valuation for discrete normal mixtures. *The journal of financial researcg 13*(4), 2850296.

Rosenberg, J. V. (1998). Pricing multivariate contingent claims using estimated risk–neutral density functions. *Journal of International Money and Finance 17*(2), 229 – 247.

Rubinstein, M. (1994). Implied binomial trees. *Journal of Finance 49*(3), 771–818.

Shimko, D. (1993). Bounds of probability. *Risk 6*, 33–37.

Skiadopulos, G. (2001, Jun). Volatility smile consistent option models: A survey. *International Journal of Theoretical and Applied Finance 04*(03), 403–437.

Stutzer, M. (1996). A simple nonparametric approach to derivative security valuation. *Journal of Finance 51*, 1633–1652.

Tieleman, T. and G. Hinton (2012). Lecture 6-5: Rmsprop, divide the gradient by a running average of its recent magnitude.

Yu Zheng, Y. Y. and B. Chen (2019). Gated deep neural networks for implied volatility surfaces.

9 Appendix

We start with the definition of the Leibniz integral rule. As a reminder, this rule is used to derive an integral with a parameter as limit. We first define our integral:

$$\int_{a(x)}^{b(x)} h(x,t)dt \tag{16}$$

Then the Leibniz integral rule states that:

$$\frac{\partial}{\partial x}\Big(\int_{a(x)}^{b(x)} h(x,t)dt\Big) = h(x,b(x))\frac{\partial}{\partial x}b(x) - h(x,a(x))\frac{\partial}{\partial x}a(x) + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x}h(x,t)dt \tag{17}$$

We use the Risk-Neutral pricing of a call option:

$$C = e^{-rT}\int_{K}^{+\infty}(S_T - K)f(S_T)dS_T \tag{18}$$

With r the risk-free rate, T the remaining time until maturity, K the strike, S_T the price of the underlying at time T, and $f(S_T)$ the probability density for the terminal value of the underlying, known as the *risk neutral density*. Using the notation of the (16), we define:

$$h(K, S_T) = (S_T - K)f(S_T) \tag{19}$$

Now we can have our first derivative of the call price, with respect to the strike K:

$$\frac{\partial C}{\partial K} = \frac{\partial}{\partial K}e^{-rT}\int_{K}^{+\infty}(S_T - K)f(S_T)dS_T$$

e^{-rT} does not depends on K:

$$\frac{\partial C}{\partial K} = e^{-rT} \frac{\partial}{\partial K} \int_K^{+\infty} (S_T - K) f(S_T) dS_T$$

We now can use the Leibniz rule (17) and our defined function (19):

$$\frac{\partial C}{\partial K} = e^{-rT} \Big[h(K, +\infty) * 0 - h(K, K) * 1 + \int_K^{+\infty} \frac{\partial}{\partial K} h(K, S_T) dS_T \Big]$$

Leading to:

$$\frac{\partial C}{\partial K} = e^{-rT} \Big[-(K - K) f(K) + \int_K^{+\infty} -f(S_T) dS_T \Big]$$

Finally:

$$\frac{\partial C}{\partial K} = -e^{-rT} \int_K^{+\infty} f(S_T) dS_T \tag{20}$$

For the second derivative, we begin by rearranging our last expression (20), since $f(S_T)$ is the RND, and its integral between 0 and $+\infty$ is equal to 1:

$$\frac{\partial C}{\partial K} = -e^{-rT} \Big(1 - \int_0^K f(S_T) dS_T \Big)$$

And:

$$-e^{rT} \frac{\partial C}{\partial K} - 1 = -\int_0^K f(S_T) dS_T$$

Finally:

$$e^{rT} \frac{\partial C}{\partial K} + 1 = \int_0^K f(S_T) dS_T \tag{21}$$

Let's work with those two expressions, and derive them with respect to the strike. Let's start with the right part:

$$\int_0^K f(S_T) dS_T \tag{22}$$

We easily can apply again the Leibniz integral rule:

$$\frac{\partial}{\partial K} \int_0^K f(S_T) dS_T = f(S_T) \frac{\partial}{\partial K} K - 0 + \int_0^K \frac{\partial}{\partial K} f(S_T) dS_T = f(S_T) \tag{23}$$

Now we work on the left side (21):

$$\frac{\partial}{\partial K} \Big(e^{rT} \frac{\partial C}{\partial K} + 1 \Big) = e^{rT} \frac{\partial^2 C}{\partial K^2} \tag{24}$$

We get our final expression of the RND:

$$f(S_T) = e^{rT} \frac{\partial^2 C}{\partial K^2} \tag{25}$$

The same result holds for puts.