

Lee, Geon; Kim, Tae-Kyoung; Kim, Hyun-Gyoon; Huh, Jeonggyu

Article

Newton-Raphson emulation network for highly efficient computation of numerous implied volatilities

Journal of Risk and Financial Management

Provided in Cooperation with:

MDPI – Multidisciplinary Digital Publishing Institute, Basel

Suggested Citation: Lee, Geon; Kim, Tae-Kyoung; Kim, Hyun-Gyoon; Huh, Jeonggyu (2022) : Newton-Raphson emulation network for highly efficient computation of numerous implied volatilities, Journal of Risk and Financial Management, ISSN 1911-8074, MDPI, Basel, Vol. 15, Iss. 12, pp. 1-8,
<https://doi.org/10.3390/jrfm15120616>

This Version is available at:

<https://hdl.handle.net/10419/275093>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.



If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.



<https://creativecommons.org/licenses/by/4.0/>

Article

Newton–Raphson Emulation Network for Highly Efficient Computation of Numerous Implied Volatilities

Geon Lee ¹, Tae-Kyoung Kim ² , Hyun-Gyoon Kim ³ and Jeonggyu Huh ^{4,*} 

¹ Department of Mathematics, Chonnam National University & Statistics, Gwangju 61186, Republic of Korea

² Asset Management Department, KB Kookmin Bank, Seoul 07328, Republic of Korea

³ Department of Mathematics, Yonsei University, Seoul 03722, Republic of Korea

⁴ Department of Statistics, Chonnam National University, Gwangju 61186, Republic of Korea

* Correspondence: huhjeonggyu@jnu.ac.kr

Abstract: In finance, implied volatility is an important indicator that reflects the market situation immediately. Many practitioners estimate volatility by using iteration methods, such as the Newton–Raphson (NR) method. However, if numerous implied volatilities must be computed frequently, the iteration methods easily reach the processing speed limit. Therefore, we emulate the NR method as a network by using PyTorch, a well-known deep learning package, and optimize the network further by using TensorRT, a package for optimizing deep learning models. Comparing the optimized emulation method with the benchmarks, implemented in two popular Python packages, we demonstrate that the emulation network is up to 1000 times faster than the benchmark functions.

Keywords: graphics processing unit (GPU) accelerated computing; implied volatility; Newton–Raphson method; PyTorch; TensorRT



Citation: Lee, Geon, Tae-Kyoung Kim, Hyun-Gyoon Kim, and Jeonggyu Huh. 2022. Newton–Raphson Emulation Network for Highly Efficient Computation of Numerous Implied Volatilities. *Journal of Risk and Financial Management* 15: 616. <https://doi.org/10.3390/jrfm15120616>

Academic Editor: David Liu

Received: 1 November 2022

Accepted: 15 December 2022

Published: 18 December 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Volatility is the degree of variability in underlying asset dynamics, helping investors predict future market variability, and is usually divided into historical and implied volatility. Because historical volatility is obtained from information for a specific period in the past, this type of volatility lags behind the market situation. Unlike historical volatility, implied volatility contains only current market information, not past market information (Gatheral 2011). When a sudden shock, such as a financial crisis, occurs, implied volatility is fairly important in predicting future volatility. See (Hull 2003; Wilmott 2013) for a detailed explanation of volatility.

When using implied volatility for various purposes, such as estimating parameters of an option pricing model, it is often necessary to convert a large number of option prices into implied volatilities in real time. However, iterative methods, such as the bisection and Newton–Raphson (NR) methods, typically used to obtain implied volatilities, are unsuitable for calculating numerous implied volatilities due to excessive computation. Therefore, many studies (Brenner and Subrahmanyam 1988; Chance 1996; Corrado and Miller 1996; Jäckel 2006; Li 2005; Mininni et al. 2021; Orlando and Tagliatalata 2017; Stefanica and Radoičić 2017) have proposed several formulas to approximate implied volatility. For example, Brenner and Subrahmanyam (1988); Chance (1996) and Li (2005) employed the Taylor expansion, Corrado and Miller (1996) utilized the quadratic approximation, and Mininni et al. (2021) used hyperbolic tangent functions to approximate the implied volatility.

However, increasing the accuracy by using various mathematical methods already faces a limitation. Thus, in line with various studies (Berg and Nyström 2019; Chen et al. 2018; Li et al. 2020; Raissi and Karniadakis 2018; Raissi et al. 2019; Ramuhalli et al. 2005) that have supplemented numerical schemes, such as the finite element method, with neural networks, deep learning methods Kim et al. (2022); Liu et al. (2019) are introduced to improve the accuracy of estimating implied volatility. As in many existing studies estimating

the implied volatility (e.g., Jäckel (2006, 2015); Kim et al. (2022)), an iteration procedure is added after the network approximation to attain higher accuracy. This additional iteration procedure significantly burdens the whole estimation process, though. In the experiments performed by Kim et al. (2022), for example, the iteration procedure takes considerably more time compared to the network approximation of implied volatility.

Thus, in this study, we are concerned with further reducing the estimation time by facilitating the iteration procedure. We develop a graphics processing unit (GPU) acceleration scheme for the NR method, reducing the computational time for estimating implied volatilities dramatically. To this end, we apply the so-called neural emulation technique, which implements an algorithm like a neural network with zero or very few parameters. This technique enables employing well-known deep learning packages, such as TensorFlow and PyTorch, to accelerate a scientific procedure. These popular packages make it straightforward to implement large-scale parallel computation by using GPUs. Additionally, this approach allows a neural network optimization engine, TensorRT, to further maximize inference performance. We refer to the network emulating the NR method as the NR emulation network, and this study has an implication for accelerating the iteration process through the NR emulation network.

The presented NR emulation network was compared with the benchmarks, implemented widely used two packages, in terms of estimation accuracy and speed to verify the effectiveness of this study. The test results reveal that the NR emulation network is up to 1000 times faster than the benchmarks of the two well-known packages, but with similar accuracy. In other words, the proposed NR emulation network is stable and efficient enough to be used in estimating numerous implied volatilities in practice.

The background, such as the implied volatility and the NR method, is provided in the next section. Section 3 fully describes the NR emulation network. The NR network is compared in terms of accuracy and computation time with the benchmarks in Section 4. The last section concludes the work.

2. Backgrounds

2.1. Implied Volatility

An option is a contract that trades the right to buy (call option) and sell (put option) an asset at a predetermined strike price on a maturity date. In addition, options can be divided into several types depending on the exercise method. If the option can be exercised only on the expiration date of the contract, it is a European-style option. The Black–Scholes model (Black and Scholes 1973) is generally used to evaluate European options.

In the Black–Scholes model, the option pricing formula is given by

$$\begin{aligned} c^{call}(S_t, t; r, \sigma, K, T) &= S_t N(d_1) - Ke^{-r(T-t)} N(d_2) \\ c^{put}(S_t, t; r, \sigma, K, T) &= Ke^{-r(T-t)} N(-d_2) - S_t N(-d_1), \end{aligned} \tag{1}$$

where S_t is the stock price at t , r denotes the risk-free rate, σ represents the volatility of S_t , K and T are the strike price and expiration time of the option, respectively, $d_1 = \frac{1}{\sigma\sqrt{T-t}} \left\{ \ln \frac{S_t}{K} + (r + \frac{1}{2}\sigma^2)(T-t) \right\}$, $d_2 = d_1 - \sigma\sqrt{T-t}$, $N(\cdot)$ denotes the cumulative distribution function of the standard normal distribution, and c^{call} and c^{put} indicates the prices for the call and put options, respectively. Among the variables that influence the option price c , except for the volatility σ , the other variables S_t , t , r , K , and T can be provided from the market information and the option specification, whereas σ must be estimated by using market data. However, in many cases, the market price c_{mkt} of the option is a known quote because most options are exchange-traded products, and the corresponding σ is reversely calculated from the price c_{mkt} . The value of σ computed in this way is called implied volatility σ_{impl} .

In other words, for a given S_t , r , t , K , T , and c_{mkt} , the implied volatility σ_{impl} is defined for each option as follows:

$$c_{mkt} = h_{r,k,\tau}(\sigma_{impl}) := c(S_t, t; r, \sigma_{impl}, K, T), \tag{2}$$

where $k = S_t / K$ and $\tau = T - t$. Because $h_{r,k,\tau}(\cdot)$ is monotonically increasing, σ_{impl} uniquely exists as $h_{r,k,\tau}^{-1}(c_{mkt})$ if c_{mkt} is within an appropriate range. In addition, σ_{impl} is often considered an alternative indicator of c_{mkt} because σ_{impl} changes in a more stable way than c_{mkt} .

2.2. Newton–Raphson Iterative Method

The nonlinear Equation (2) must be solved with a numerical scheme for determining σ_{impl} because $h_{r,k,\tau}^{-1}$ is not found explicitly. An iterative method, such as the bisection or secant method, is commonly used to determine a solution to the nonlinear equation. In particular, the NR method, which is an algorithm with a fast convergence rate, is most used for estimating σ_{impl} .

According to the NR method, the implied volatility σ_{impl} can be obtained in a series of the following update steps:

$$\sigma_{n+1} = \sigma_n - \frac{h_{r,k,\tau}(\sigma_n) - c_{mkt}}{h'_{r,k,\tau}(\sigma_n)} \tag{3}$$

If the initial value σ_0 is given within the convergence interval, the NR method converges rapidly to σ_{impl} with a quadratic convergence rate. However, there is a risk of divergence if σ_0 is not given in the convergence interval. Fortunately, the convergence of the NR method is guaranteed if σ_0 is set to σ_c , as follows (refer to Higham 2004),

$$\sigma_c = \sqrt{\left| \frac{2}{\tau} (\ln k + r\tau) \right|}, \tag{4}$$

where σ_c is the unique inflection point of $h_{r,k,\tau}$, where the option vomma is 0. The first and second derivatives $\frac{\partial c}{\partial \sigma}$ and $\frac{\partial^2 c}{\partial \sigma^2}$ of the option price c with respect to σ are called vega v and vomma v' , respectively.

3. Newton–Raphson Emulation Network

This section proposes and describes the NR emulation network emulating the NR method. The emulation network enables us to obtain numerous implied volatilities in real time through parallel computing of the GPU and optimizing the computation graphs of the network.

The NR update (NRU) layer depicted in Figure 1 is designed to emulate the update step (3) of the NR method. In addition, $h_{r,k,\tau}$ of the NRU layer is defined in (2). Therefore, if the input σ_n passes through the NRU layer, one step of the NR method is applied to produce σ_{n+1} , which is expected to be closer to σ_{impl} than σ_n . Additionally, $h_{r,k,\tau}$ and $h'_{r,k,\tau}$ depend on the risk-free rate r , the ratio k of the stock price to the strike price, and the time to maturity τ ; thus, the NRU layer also depends on r , k , and τ . In addition, the NRU layer is also dependent on c_{mkt} . This dependence can also be considered for the NRU layer to be conditioned on r , k , τ , and c_{mkt} , similar to the conditional generative adversarial network (Mirza and Osindero 2014).

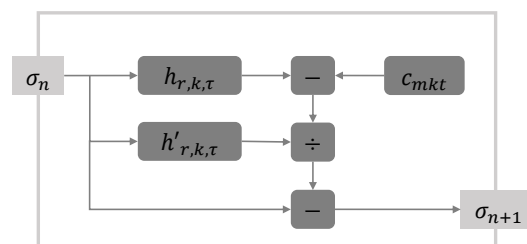


Figure 1. Newton–Raphson update layer.

The NR emulation network is created by stacking NRU layers as depicted in Figure 2, which corresponds to the process of repeating the update steps of the NR method. As the input σ_0 for the network, σ_c in Equation (4) is chosen. This choice ensures that the output σ_{pred} is sufficiently close to σ_{impl} if the emulation network is deep enough (except in the cases where a too-small σ_{impl} makes σ_{pred} diverge because of the limitations of the floating point number system). Passing through the deep network means performing the update steps of the NR method many times. In the experiments that follow in the next section, it is empirically demonstrated that the minimum depth of the NR emulation network should be eight to guarantee convergence. In other words, when $\sigma_0 = \sigma_c$, there should be at least eight NRU layers in the network such that $|\sigma_{pred} - \sigma_{impl}| < \epsilon$ for the machine epsilon $\epsilon (\approx 10^{-6})$ of the single-precision floating system.

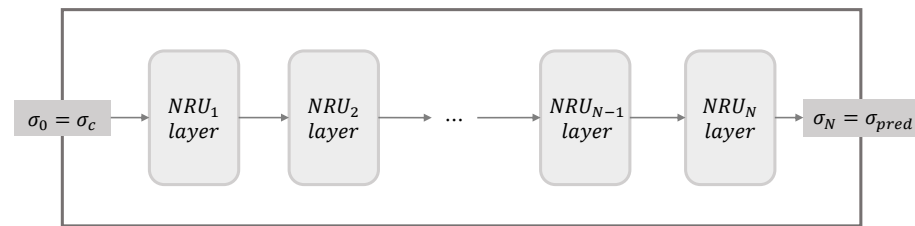


Figure 2. Newton–Raphson (NR) emulation network.

To exploit powerful parallel computing, we implement the NR emulation network with PyTorch, a well-known deep learning framework, and run it on the GPU. This approach also allows for optimizing the network with TensorRT to accelerate the inference performance of the emulation network. In addition, TensorRT is one of the deep learning-related tools provided by NVIDIA, which can be used to optimize the structure of a network while converting a dynamic graph of PyTorch into a static graph (<https://developer.nvidia.com/tensorrt>, accessed on 1 November 2022). Although it is usual to use neural networks to identify patterns inherent in data, such a data-learning stage does not exist in this study.

In the next section, we experimentally reveal how accurately and quickly the NR emulation network derives the implied volatility. We conclude that market prices c_{mkt} of numerous options can be converted into implied volatilities σ_{impl} in real time.

4. Numerical Tests

4.1. Test Data Description

A testing dataset with one million data points was prepared by generating virtual option prices c_{mkt} by using the Black–Scholes Formula (1), converting them to the corresponding implied volatilities σ_{impl} . The variables σ , τ , and k involved in generating c_{mkt} are randomly selected within the ranges as in Table 1 (for convenience, the risk-free rate r is fixed to 0 to offset its effect). The variables σ , τ , and k are the volatility parameter for the Black–Scholes model, time to maturity $T - t$, and ratio S_t/K of the stock price to exercise price, respectively.

Table 1. Variable ranges involved in generating the virtual test data. $U(a, b)$ is the uniform distribution on (a, b) .

| Variable | σ_{impl} | τ | $\ln k$ |
|--------------|-----------------|--------------|---|
| Distribution | $U(0.01, 0.5)$ | $U(0.01, 2)$ | $U(-\frac{\sigma^2}{2}\tau - 2\sigma\sqrt{\tau}, -\frac{\sigma^2}{2}\tau + 2\sigma\sqrt{\tau})$ |

We set the variable ranges to be as acceptable as possible by considering and reflecting the real market. Most options in the real market have a time to maturity τ of less than two years, and typically, the volatility σ does not fall below 1% and does not exceed 50%.

Moreover, the strike price k is set to be within the 95% confidence interval of the distribution of the stock price S_τ at time τ , and the distribution is obtained from the assumption of Black and Scholes that $\ln S_\tau$ follows $N(-\frac{\sigma^2}{2}\tau, \sigma^2\tau)$ when $S_0 = 1$ and $r = 0$.

4.2. Test Results

In this section, we analyze the results of various tests. As benchmarks, we choose the NR method from the Python package SciPy (<https://scipy.org/>) (accessed on 1 November 2022) and the implied volatility estimation function from the recently released Python package py_vollib_vectorized (https://github.com/marcdemers/py_vollib_vectorized) (accessed on 15 November 2022). We denote the methods of SciPy and py_vollib_vectorized as SciPy-NR and Vectorized, respectively.

The SciPy-NR method is set to estimate the implied volatility through eight iterations, so the emulation network is also set to perform the estimation through eight NRU layers. Eight is the minimum number for both methods to reduce errors to near the machine epsilon ϵ ($\approx 10^{-6}$) of the single-precision floating number system.

The Python package py_vollib_vectorized was released in 2021, and is the latest among the packages for estimating implied volatility. The package is theoretically based on Jäckel (2015), and works in parallel on CPU to run faster.

Table 2 reveals that the NR emulation network runs on the GPU to take full advantage of parallel computing. However, except for expensive Tesla GPUs, ordinary GPUs specialize in single-precision floating numbers, not double precision. In this study, we do not have a Tesla GPU; thus, we process the tests based on the single-precision floating number system. Therefore, for a fair comparison, the benchmark methods are also conducted with the precision of the single-precision floating numbers.

Table 2. Implementation platform and hardware.

| | SciPy-NR & Vectorized | NR Emulation |
|----------|------------------------------|-------------------------------|
| Platform | SciPy (Python) | PyTorch + TensorRT (Python) |
| Hardware | CPU (Intel Xeon Silver 4216) | GPU (NVIDIA GeForce RTX 2080) |

Table 3 compares the accuracy of each method by using the mean absolute error (MAE), mean square error (MSE), and mean relative error (MRE) for inferring the implied volatility. The definitions of MAE, MSE, and MRE are provided as follows:

$$MAE = \frac{1}{L} \sum_{i=1}^L |\sigma_{i,pred} - \sigma_{i,impl}|, \quad MSE = \frac{1}{L} \sum_{i=1}^L (\sigma_{i,pred} - \sigma_{i,impl})^2, \quad MRE = \frac{1}{L} \sum_{i=1}^L \frac{|\sigma_{i,pred} - \sigma_{i,impl}|}{\sigma_{i,impl}},$$

where $L = 1,000,000$, and $\sigma_{i,pred}$ denotes the value derived by the emulation network to predict $\sigma_{i,impl}$. Both methods achieve the maximum possible accuracy on the single-precision floating number system, as the values of MAE and MRE are below ϵ , and the value of MSE is below ϵ^2 . The SciPy-NR and Vectorized methods tend to infer σ_{impl} 10 times more accurately than the emulation network, implying that the CPU may achieve higher precision than the GPU, even if both processing units work on similar single-precision floating number systems.

Table 3. Implied volatility estimation error.

| Error Type | SciPy-NR | Vectorized | NR Emulation |
|------------|----------------------------|----------------------------|----------------------------|
| MAE | 2.800171×10^{-8} | 2.890932×10^{-8} | 2.816055×10^{-7} |
| MSE | 1.930116×10^{-15} | 1.988556×10^{-15} | 2.949284×10^{-13} |
| MRE | 2.155739×10^{-7} | 2.184584×10^{-7} | 1.962279×10^{-6} |

Table 4 presents the computation time consumed for the execution of each method. The NR emulation network has a very short computation time compared to the SciPy-NR and Vectorized methods. Additionally, as the number of implied volatility estimates increases, the emulation network becomes overwhelmed by the SciPy-NR and Vectorized methods in terms of processing speed. When the number of implied volatility estimates reaches one million, the running time of the network is about 1000 times shorter than that of the SciPy-NR method and about 700 times shorter than Vectorized. The computation times are repeatably measured 100 times, and the average and standard deviation of the resultant values are written together.

Table 4. Computation times (in milliseconds) for estimating the implied volatility. Each value is calculated by averaging the values from 100 repetitions, and the corresponding standard deviation is provided in parentheses.

| # of Implied Volatility Estimates | SciPy-NR | Vectorized | NR Emulation |
|-----------------------------------|-------------------|------------------|---------------|
| 10,000 | 14.71 (3.0527) | 7.22 (0.5959) | 0.4 (0.0182) |
| 100,000 | 99.07 (4.7911) | 72.04 (6.5210) | 0.44 (0.01) |
| 1,000,000 | 1212.64 (11.2775) | 754.19 (31.7333) | 1.50 (0.0065) |

Figure 3 depicts how the MSE changes each time it passes through the NRU layer of the NR emulation network. The MSE decreases by about 10^{-1} from the first to third NRU layers and by about 10^{-2} from the fourth to sixth NRU layers. In contrast, the seventh and eighth NRU layers reduce the MSE only slightly because the sixth layer has already virtually achieved the maximum possible accuracy of the single-precision floating number system.

Lastly, we demonstrate how the inference value σ_{pred} changes while passing through the NRU layers. Table 5 presents two specific cases: (1) $\sigma = 0.3$, $\tau = 1$, and $k = 1.5$ and (2) $\sigma = 0.3$, $\tau = 1$, and $k = 1.3$. The emulation network produces virtually the exact outputs at the seventh and fourth layers for $k = 1.5$ and $k = 1.3$, respectively. The outputs are indistinguishable from the exact implied volatility σ_{impl} on the single-precision floating number system. These results confirm that the number of NRU layers required to achieve accurate implied volatility differs individually depending on the option.

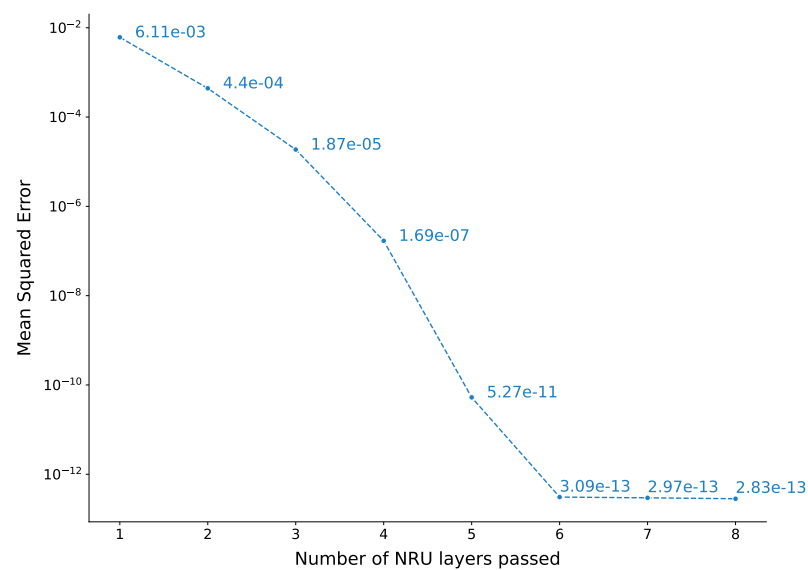


Figure 3. The degree of MSE change according to the number of NRU layers passed.

Table 5. Change in the predicted value of the NR neural network according to the number of NRU layers passed.

| # of NRU Layers Passed | $\sigma = 0.3, \tau = 1$ | |
|------------------------|--------------------------|------------------|
| | $k = 1.5$ | $k = 1.3$ |
| 0 | 0.90051656961441 | 0.72438144683838 |
| 1 | 0.37598699331284 | 0.32452529668808 |
| 2 | 0.30990260839462 | 0.30062055587769 |
| 3 | 0.30027109384537 | 0.30000048875809 |
| 4 | 0.30000036954880 | 0.30000001192093 |
| 5 | 0.30000007152557 | 0.30000001192093 |
| 6 | 0.30000016093254 | 0.30000001192093 |
| 7 | 0.30000001192093 | 0.30000001192093 |
| 8 | 0.30000001192093 | 0.30000001192093 |

5. Conclusions

Implied volatility is critical indicator that reflects expectations about future volatility and can be obtained by solving a nonlinear equation by using the NR method. However, it is often necessary to repeatedly estimate numerous implied volatilities. The iterative method then fails because of a heavy computational burden. Therefore, the NR emulation network is proposed in this study to resolve the challenge. To develop the network, we implemented the NR method, like a PyTorch network, and optimized the network with TensorRT. As a result, the emulation network is up to 1000 times faster than the well-known benchmarks, showing that the proposed method is stable and efficient enough to be utilized for computation of numerous implied volatilities in practice.

The purpose of this work is achieved by emulating and optimizing the NR method without taking a complex mathematical approach, which is a distinctive contribution to literature compared to the existing results with complicated techniques. Moreover, all codes are uploaded online for the NR emulation network to be utilized efficiently (<https://github.com/thix-is/Newton-Raphson-emulation>, accessed on 11 November 2022). Our result implies the possibility to contribute to solving other difficult issues of computational finance, such as model parameter calibration, due to the recent progress in computing technology. Therefore, follow-up studies are required to address these problems using the neural emulation technique.

Author Contributions: Conceptualization, G.L. and J.H.; methodology, G.L.; software, G.L. and T.-K.K.; validation, H.-G.K.; formal analysis, G.L. and H.-G.K.; investigation, G.L.; resources, G.L. and H.-G.K.; data curation, G.L. and T.-K.K.; writing—original draft preparation, G.L.; writing—review and editing, J.H.; visualization, G.L.; supervision, J.H.; project administration, J.H.; funding acquisition, J.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the BK21 Fostering Outstanding Universities for Research (No. 5120200913674) funded by the Ministry of Education (Korea) and the National Research Foundation of Korea. Jeonggyu Huh received financial support from the National Research Foundation of Korea (Grant No. NRF-2022R1F1A1063371). This work was supported by the artificial intelligence industrial convergence cluster development project funded by the Ministry of Science and ICT (Korea) and Gwangju Metropolitan City.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Berg, Jens, and Kaj Nyström. 2019. Data-driven discovery of pdes in complex datasets. *Journal of Computational Physics* 384: 239–52. [CrossRef]
- Black, Fischer, and Myron Scholes. 1973. The pricing of options and corporate liabilities. *Journal of Political Economy* 81: 637–54. [CrossRef]
- Brenner, Menachem, and Marti G Subrahmanyam. 1988. A simple formula to compute the implied standard deviation. *Financial Analysts Journal* 44: 80–83. [CrossRef]
- Chance, Don M. 1996. A generalized simple formula to compute the implied volatility. *Financial Review* 31: 859–67. [CrossRef]
- Chen, Ricky T. Q., Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. 2018. Neural Ordinary Differential Equations, in 'Advances in Neural Information Processing Systems'. La Jolla. Available online: <https://papers.nips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html> (accessed on 10 November 2022).
- Corrado, Charles J., and Thomas W. Miller Jr. 1996. A note on a simple, accurate formula to compute implied standard deviations. *Journal of Banking & Finance* 20: 595–603.
- Gatheral, Jim. 2011. *The Volatility Surface: A Practitioner's Guide*. Oxford: John Wiley & Sons.
- Higham, Desmond J. 2004. *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*. Cambridge: Cambridge University Press.
- Hull, John C. 2003. *Options Futures and Other Derivatives*. Noida: Pearson Education India.
- Jäckel, Peter. 2006. By implication. *Wilmott* 26: 60–66.
- Jäckel, Peter. 2015. Let's be rational. *Wilmott* 2015: 40–53. [CrossRef]
- Kim, Tae-Kyoung, Hyun-Gyoon Kim, and Jeonggyu Huh. 2022. Large-scale online learning of implied volatilities. *Expert Systems with Applications* 203: 117365. [CrossRef]
- Li, Steven. 2005. A new formula for computing implied volatility. *Applied Mathematics and Computation* 170: 611–25. [CrossRef]
- Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2020. Fourier neural operator for parametric partial differential equations. *arXiv* arXiv:2010.08895.
- Liu, Shuaiqiang, Cornelis W. Oosterlee, and Sander M. Bohte. 2019. Pricing options and computing implied volatilities using neural networks. *Risks* 7: 16. [CrossRef]
- Mininni, Michele, Giuseppe Orlando, and Giovanni Tagliatalata. 2021. Challenges in approximating the black and scholes call formula with hyperbolic tangents. *Decisions in Economics and Finance* 44: 73–100. [CrossRef]
- Mirza, Mehdi, and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv* arXiv:1411.1784.
- Orlando, Giuseppe, and Giovanni Tagliatalata. 2017. A review on implied volatility calculation. *Journal of Computational and Applied Mathematics* 320: 202–20. [CrossRef]
- Raissi, Maziar, and George Em Karniadakis. 2018. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics* 357: 125–41. [CrossRef]
- Raissi, Maziar, Paris Perdikaris, and George E. Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 378: 686–707. [CrossRef]
- Ramuhalli, Pradeep, Lalita Udpa, and Satish S. Udpa. 2005. Finite-element neural networks for solving differential equations. *IEEE Transactions on Neural Networks* 16: 1381–92. [CrossRef] [PubMed]
- Stefanica, Dan, and Radoš Radoičić. 2017. An explicit implied volatility formula. *International Journal of Theoretical and Applied Finance* 20: 1750048. [CrossRef]
- Wilmott, Paul. 2013. *Paul Wilmott on Quantitative Finance*. Oxford: John Wiley & Sons.