Swaption Pricing

A neural network approach

ING Belgium

TABLE OF CONTENTS

Motivation

- A classic problem in financial mathematics is the calibration of an asset pricing model to observed market prices.
- This is done by finding the parameters that make the model and market prices similar.
- In practice, one has to compute the model prices for several assets and this, for several iterations of the minimization algorithm.
- Depending on the model used, this procedure can be very time consuming as there may not be an analytical formula. In this case, simulation techniques must be used.
- In order to reduce the calibration time, we replace the model computations by an estimator that is much faster. The approximator is a deep neural network.

Methodology

- In a offline procedure, for a given pricing model (HW1F), we compute 100.000 volatility surfaces implied by the model for a given yield curve and model parameters. We then "learn" the relationship between the yield curve, model parameters and the volatility surface.
- We can then use this deterministic function approximator to compute fast volatility surfaces in the calibration procedure.

Supervised Learning

- We consider a dataset $\mathcal{D}$ of examples $\{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$ where $\boldsymbol{x}_n$ is the n^{th} input vector of dimension D. This vector contains the yield curve, the hw1f volatility parameter and the swap tenor. The output vector $\boldsymbol{y}_n$ of dimension J is the n^{th} grid of implied volatilities for 9 strikes and 8 option maturities. The number of pairs N is 100.000.
- The goal of supervised learning is to use this dataset to produce a model that takes a new vector $\boldsymbol{x}$ as input and returns a prediction of the output vector $\boldsymbol{y}$ based on the relationship that exists between the two.
- We need to find a function that approximates the output "well enough" given new inputs.

$$\boldsymbol{y}_n \approx f(\boldsymbol{x}_n), \forall n$$

Neural Network

- As predictor f, we choose to use a fully connected feedforward neural network with 4 hidden layers (L) and 32 nodes (K) per layer.
- The output at the node k in layer l is given by:

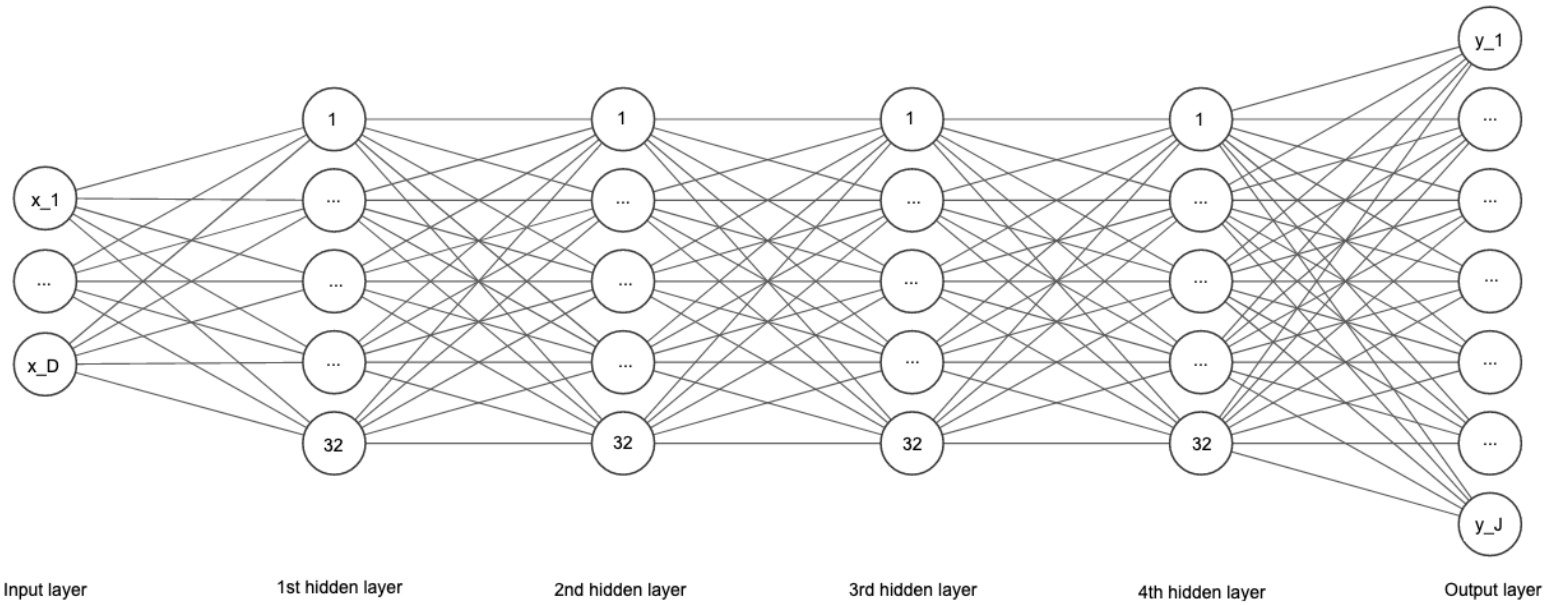
$$x_k^{(l)} = \phi\left(\sum_i w_{i,k}^{(l)} x_i^{(l-1)} + b_k^{(l)}\right)$$

  where $w_{i,k}^{(l)}$ is the edge from node i in layer $l-1$ to the node k in layer l.
- The function $\phi(.)$ is the activation function:

$$\phi(z) = \begin{cases} z & \text{if } z > 0, \\ \alpha(e^z - 1) & \text{otherwise.} \end{cases}$$

  For this application we use the ELU activation function with $\alpha = 1$.
- This structure can approximate any continuous function arbitrarily closely so long as we allow K to be large enough.

Neural Network (2)



Neural Network (3)

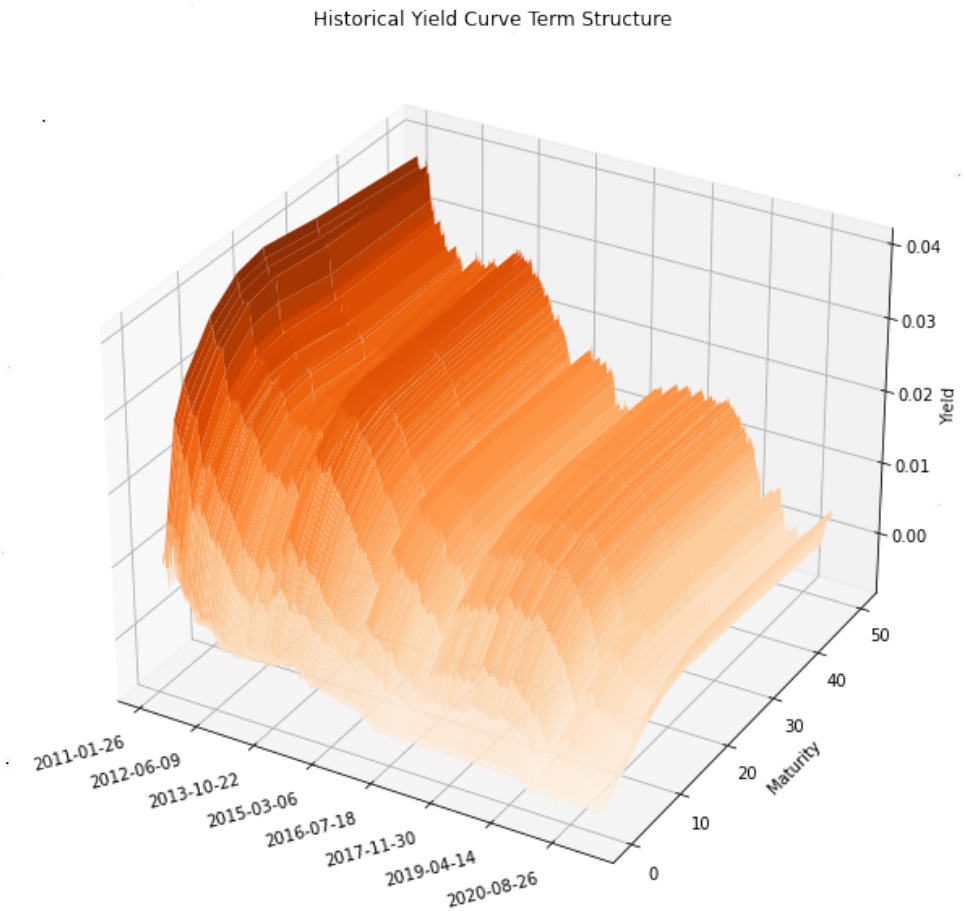
- Let $f(\boldsymbol{x})$ be the output of the neural network. We consider the Root Mean Square Error (RMSE) loss function

$$\mathcal{L} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}[\boldsymbol{y}_n - f(\boldsymbol{x}_n)]^2}$$

  as the measure of how well the NN approximates the grid of implied volatilities of the hw1f model.
- We choose the parameters of the NN w and b in order to minimize the loss function. This is done with the Adam algorithm that is an extension of Stochastic Gradient Descent.
- The neural network is implemented with the Keras library on python.

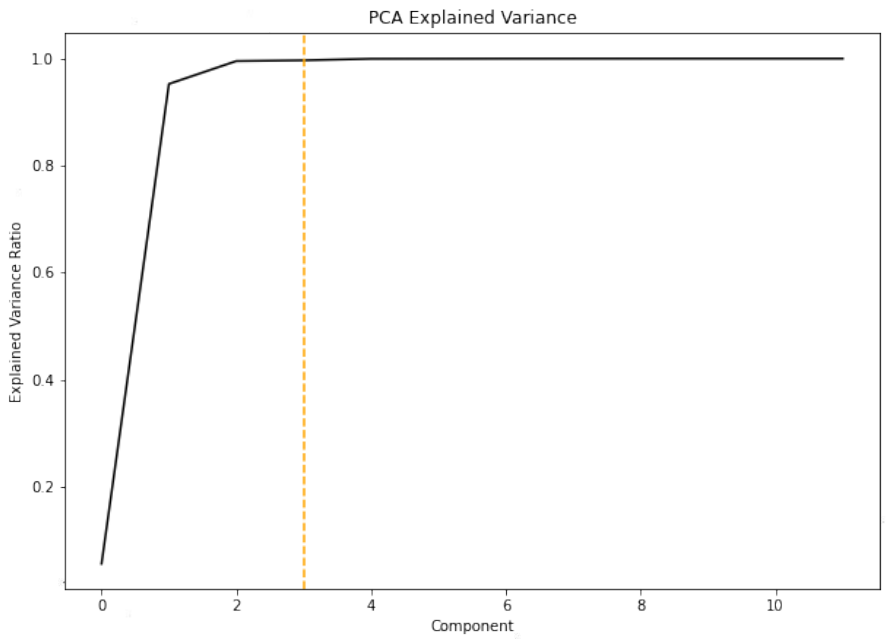
Generating Data: Input Vector

- We use 10 years of historical ESTR curves downloaded from Bloomberg. We sample these historical curves with replacement to reach 100.000 examples.



Generating Data: Input Vector (2)

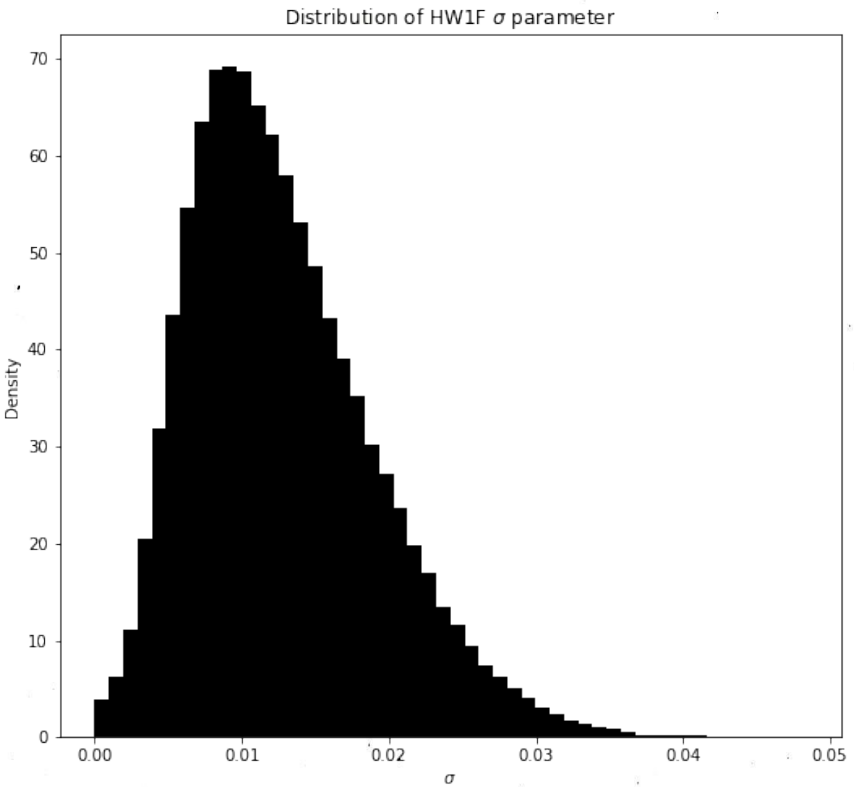
- In order to reduce the dimensionality of the problem and eliminate redundant variables from the curve, we perform a principal component analysis.

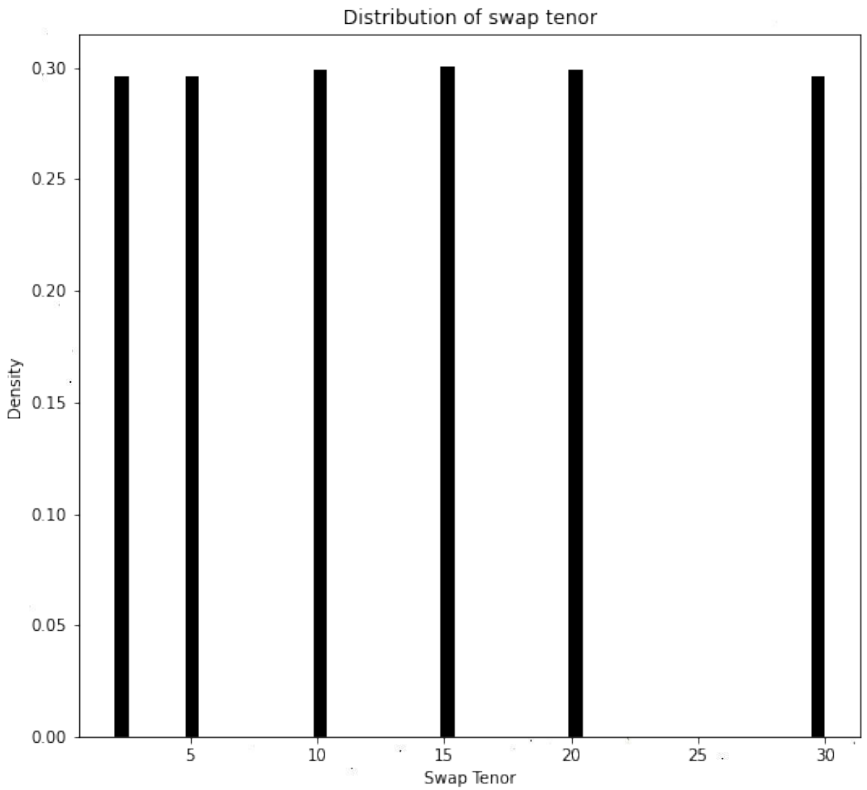


- We retain 99.9% of the variance of the curve by only considering the first three principal components. We only include these in the input vector.

Generating Data: Input Vector (3)

- The hw1f volatility parameter is sampled from a normal skew distribution with a mean of 0.5% and a variance of 1%.
- The tenor of the swap is sampled uniformly over discrete values ranging from 2 to 30 years.





Generating Data: Output Vector

- For each input vector we generate a grid of implied volatilities for 9 strikes and 8 option maturities.
- Δ strike = {-0.02, -0.015, -0.01, -0.005, 0, 0.005, 0.01, 0.015, 0.02}. This yields: $strike = swapRate + \Delta strike$.
- Option maturity = {180/360, 1, 2, 5, 10, 15, 20, 30}

Generating Data: Transformations

- The input variables are normalized between -1 and 1:

$$\tilde{x} = 2 \times \frac{x - min(x)}{max(x) - min(x)} - 1$$

  where min(X) and max(X) are the minimum and maximum values of the training set.

- The implied volatility grid is standardized:

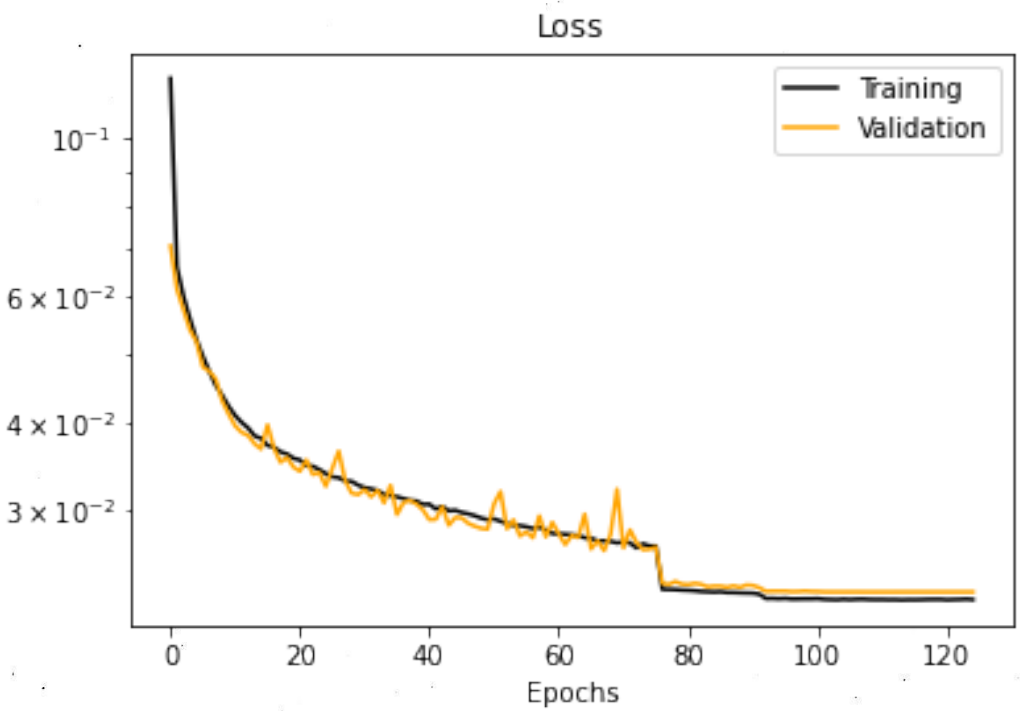
$$\tilde{y} = \frac{y - mean(y)}{std(y)}$$

  where mean(y) and std(y) are the mean and standard deviation of the training set.

- Summary: We have 100.000 samples of a pairing $(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n)$ where the input vector has 5 dimensions (first 3 principal components of the yield curve, hw1f vol, swap tenor) and the output vector has 72 dimensions (combinations of strike and option maturity).

Training the Neural Network

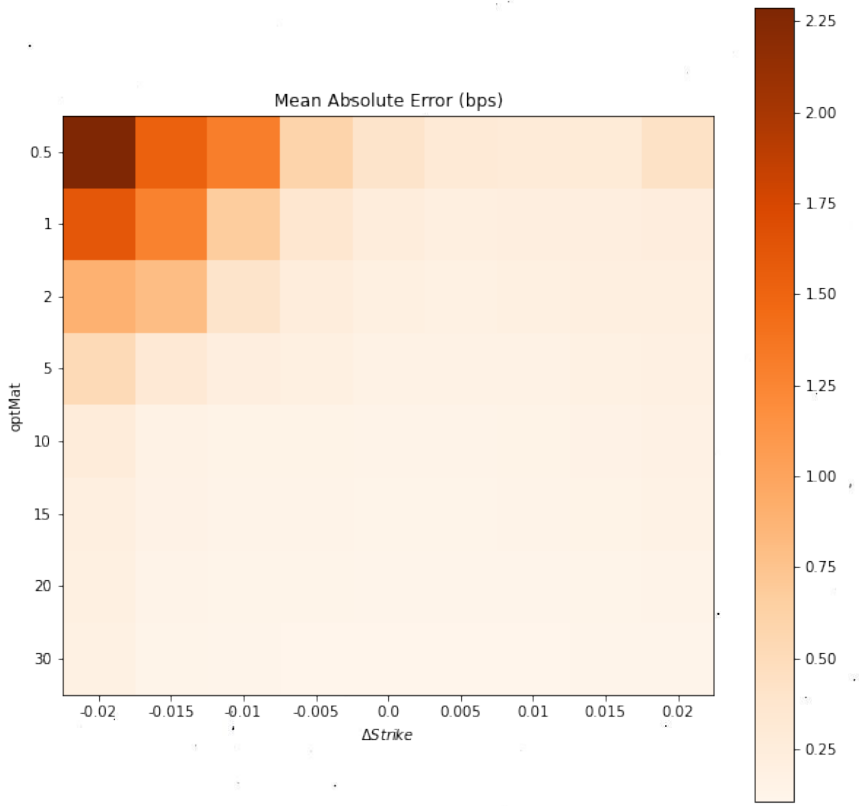
- To find the weights, we split the data into 2 parts, 64.000 samples are used to train the network and 16.000 samples are used to track the validation error while training. We stop training the NN when the validation error stops improving.
- The final 20.000 samples are held-out and only used to report the performance of the model.



- We converge after 125 epochs. The training and validation error are very similar.

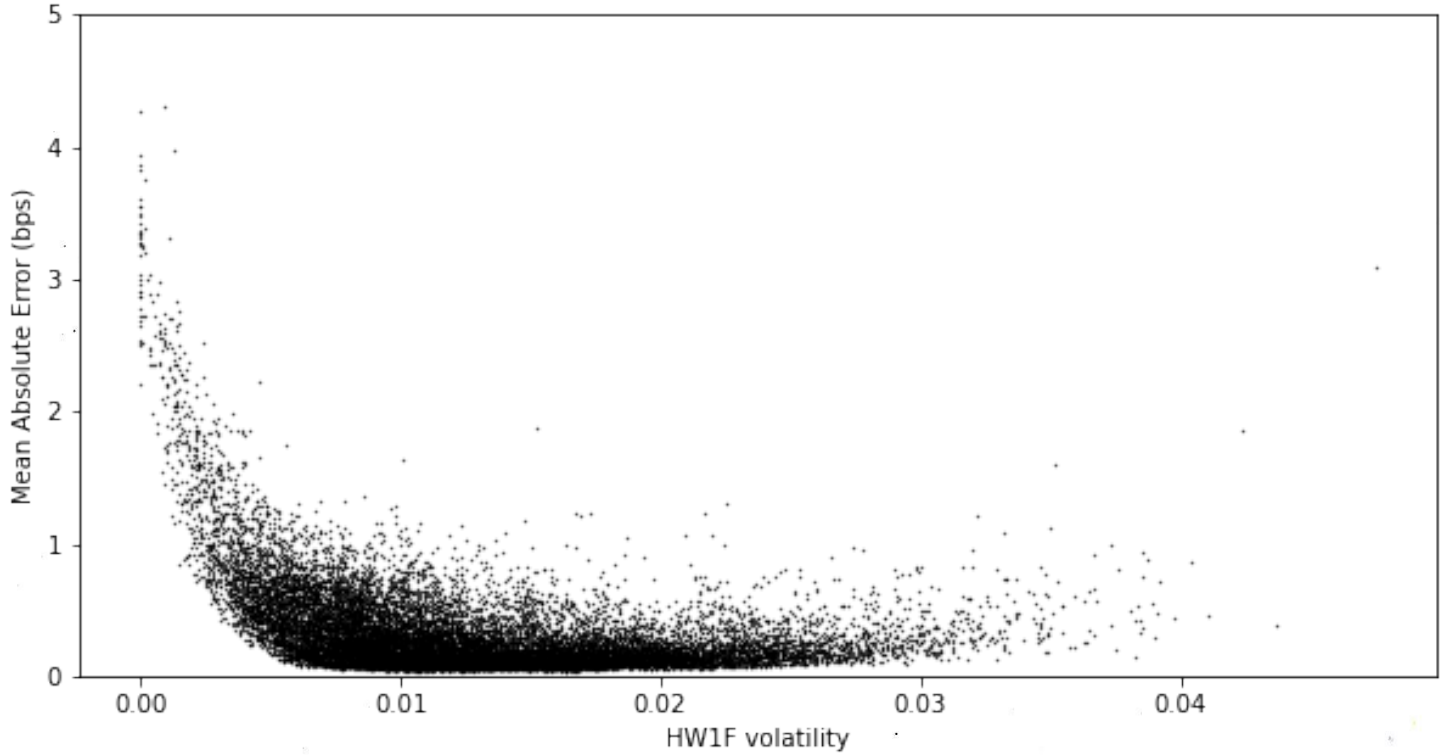
Results - Out of sample

- It appears that the NN commits on average higher absolute errors for options that are deeply in the money and have a short maturity.
- For the other combinations of strike and maturity of the option, the absolute error is less than 0.5 bps.



Results (2)

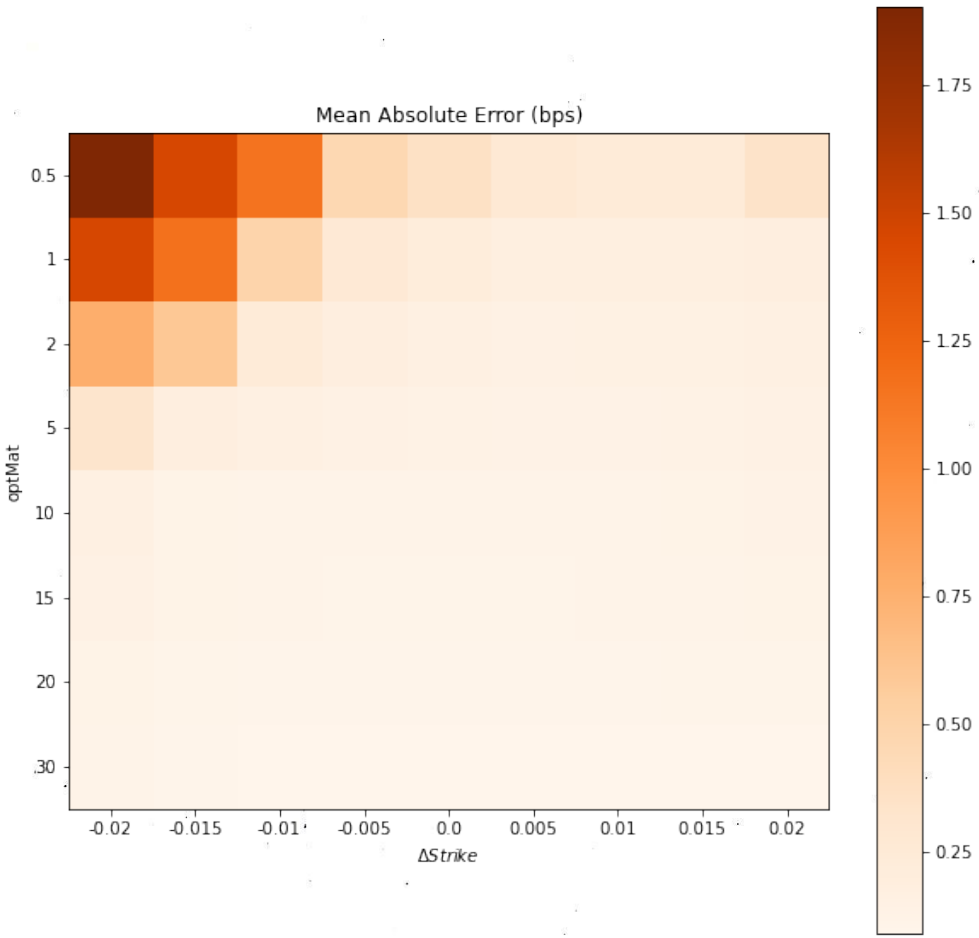
- We plot the mean absolute error of the NN for different values of hw1f volatilities.



- In general, the NN makes smaller errors for hw1f volatilities between 0.005 and 0.02. In practice, these values are much more realistic.

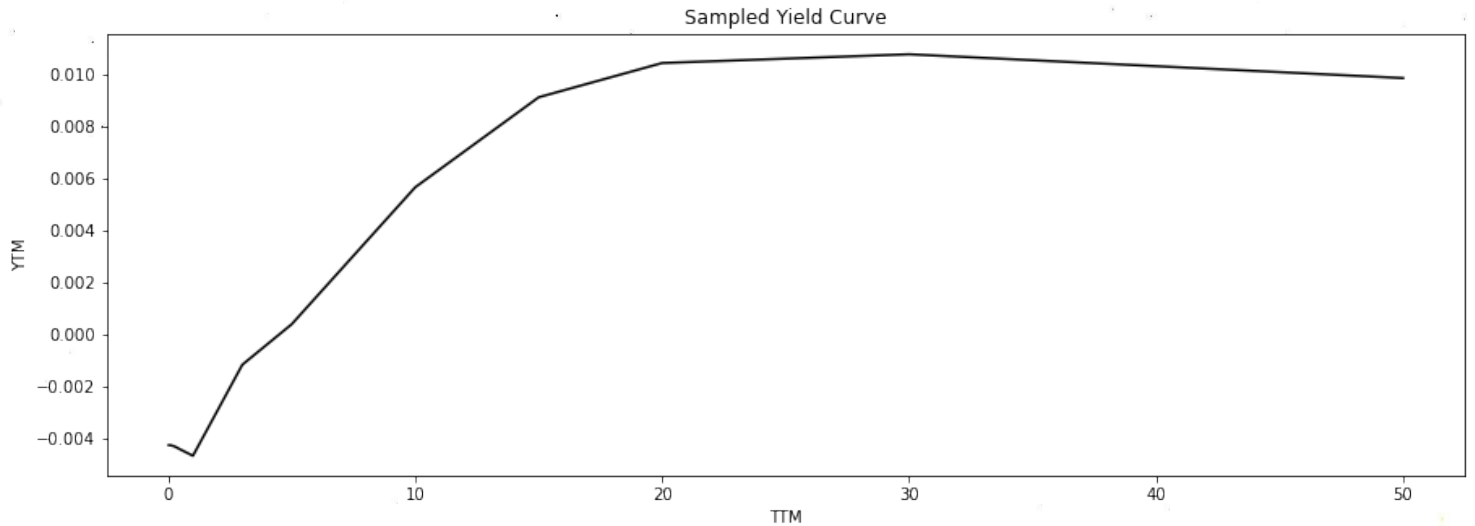
Results (3)

- If we only consider values of hw1f volatilities that are within this range in the testing set, we observe smaller mean absolute errors.



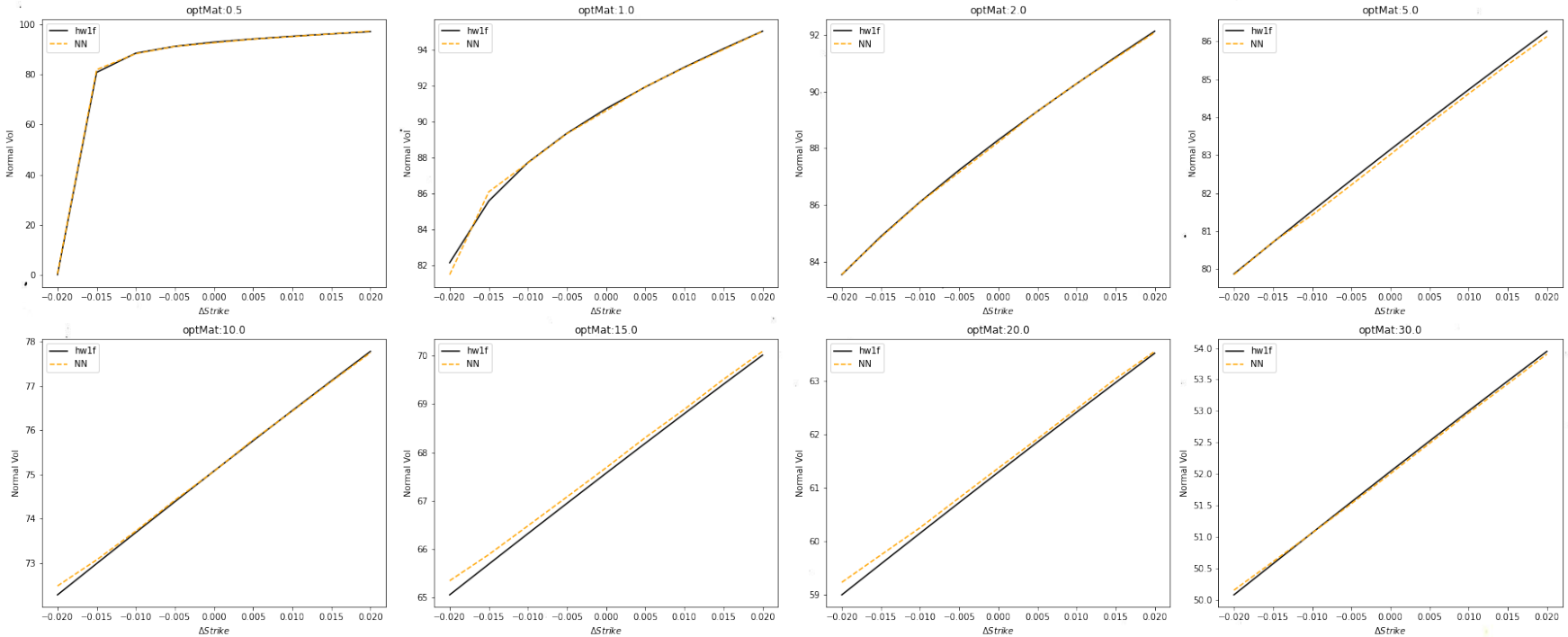
Result (4)

- We randomly select an input sample from our test data and make a prediction of the volatility surface with the NN.
- The sample in question has a hw1f volatility parameter of 0.015 and a swap tenor of 20 years. The sample curve is shown below:



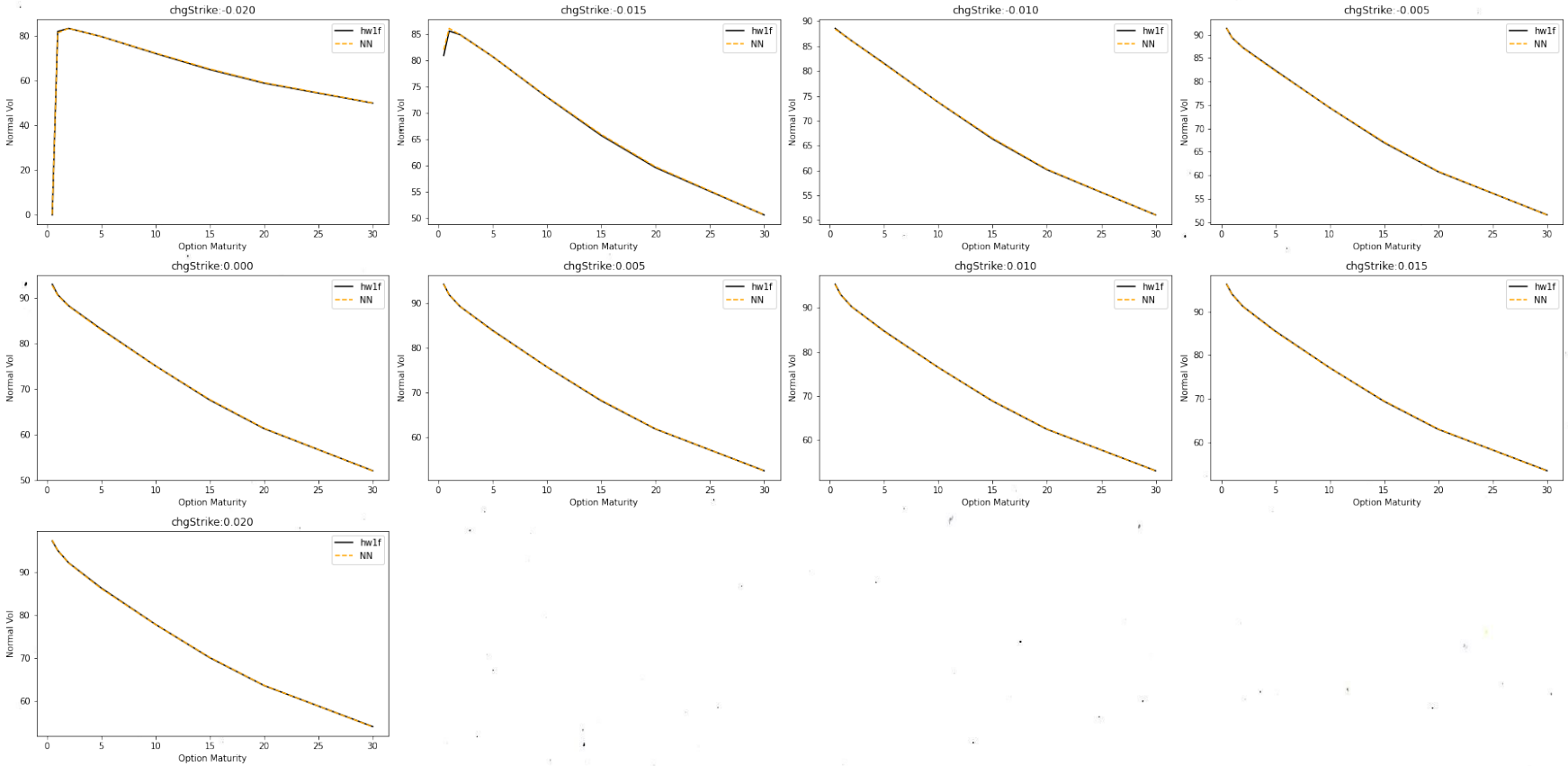
Result (5)

- For the sample, we fix the option's maturity and observe the smile implied by the NN and hw1f.



Result (5)

- For the sample, we fix the strike and observe the time depency for the NN and hw1f



Conclusion

- With the NN, we are able to calculate 100,000 implied volatility surfaces in 1.47 s ± 192 ms.
- For more complex models, the NN calculation time will be the same. Compared to pricing with a monte carlo, we get an acceleration of several orders of magnitude.
- The presented architecture is very flexible and can be adapted to support many other products and models. This would allow the use of more realistic models that are currently not feasible due to the computational time.
- The price to pay for this time saving is accuracy. However, the error seems very reasonable for a majority of strike and maturity combinations.