# Hedging a Portfolio with Greek Constraints

Machine Learning Project Report

Teo BOURSCHEIDT
Hugo BOUTEVILLAIN
Alexei CAMINADE

*ESILV A4 - Major IF (Financial Engineering)*

December 9th 2025

**Abstract**

This report details the implementation of a machine learning-based approach to hedge a portfolio of derivatives. Moving beyond traditional analytical models like Black-Scholes, we explore how supervised learning algorithms—ranging from linear regression to ensemble methods optimized via GridSearch—can predict optimal hedge ratios while strictly adhering to Greek risk constraints (Delta, Gamma, Vega).

# Contents

# 1 Introduction

## 1.1 Context

Portfolio hedging remains one of the most critical challenges in quantitative finance. Traders and asset managers must constantly manage portfolios of derivatives whose values fluctuate based on multiple risk factors, such as the underlying asset price ($S_t$), volatility ($\sigma$), and time to maturity.

Traditionally, hedging relies on analytical closed-form solutions like the Black-Scholes model [1]. However, these models rely on strong assumptions (constant volatility, frictionless markets) that rarely hold in reality. When market conditions deviate from these theoretical assumptions, or when payoffs become complex, analytical hedging becomes inaccurate [3]. This discrepancy motivates our shift towards **Machine Learning (ML)** methods to learn hedging strategies directly from data.

## 1.2 Objectives

Our goal is not simply to predict the price of an option, but to develop a model capable of:

- Estimating the optimal hedge ratio ($H_t$).
- Minimizing the portfolio's exposure to second-order risks (Greeks).
- Strictly respecting predefined risk limits imposed by risk management desks.

We trained our strategy on data generated via Monte Carlo simulations to capture realistic market dynamics.

## 1.3 Link to our specialization

As engineering students majoring in Financial Engineering, this project bridges the gap between theory and practice. It directly applies concepts from:

- **Derivatives Pricing:** Understanding the non-linear behavior of options.
- **Risk Management:** Controlling Greeks (Delta, Gamma, Vega).
- **Machine Learning:** Applying regression and optimization techniques to financial time series.

# 2 Dataset description

## 2.1 Source of the Data

To ensure we had full control over the data generation process, we created a synthetic dataset using **Monte Carlo simulations** implemented in Python [4]. This allowed us

to generate 10,000 paths including:

- Stochastic volatility components.
- Corresponding option prices and their analytical Greeks (Delta, Gamma, Vega).
- The target variable: the optimal hedge ratio derived from numerical optimization.

### 2.2 Structure of the Dataset

The resulting dataset is tabular, where each row represents a time step $t$ in a simulation. The key features used for training are summarized in Table 1.

| **Feature** | **Description** |
|---|---|
| $S_t$ | Underlying price at time $t$ |
| $r_t$ | Risk-free rate |
| $T_{min}, T_{max}, T_{mean}$ | Maturity statistics of the options in the portfolio |
| $\sigma$ | Instantaneous volatility |
| $\Delta_{i,t}, \Gamma_{i,t}, \nu_{i,t}$ | Portfolio Greeks (Delta, Gamma, Vega) |
| $\Delta_i, \Gamma_i, \nu_i$ | Option Greeks for each option i (Delta, Gamma, Vega) |
| $x_1$,...,$x_n$ | Target: Optimal positions |

Table 1: Dataset features generated via Monte Carlo simulation.

## 3 Data exploration

### 3.1 Statistics

Visual inspection of the simulated paths confirms that our data exhibits realistic financial characteristics. As shown in Figure 1, the portfolio value follows a stochastic process consistent with market behavior.

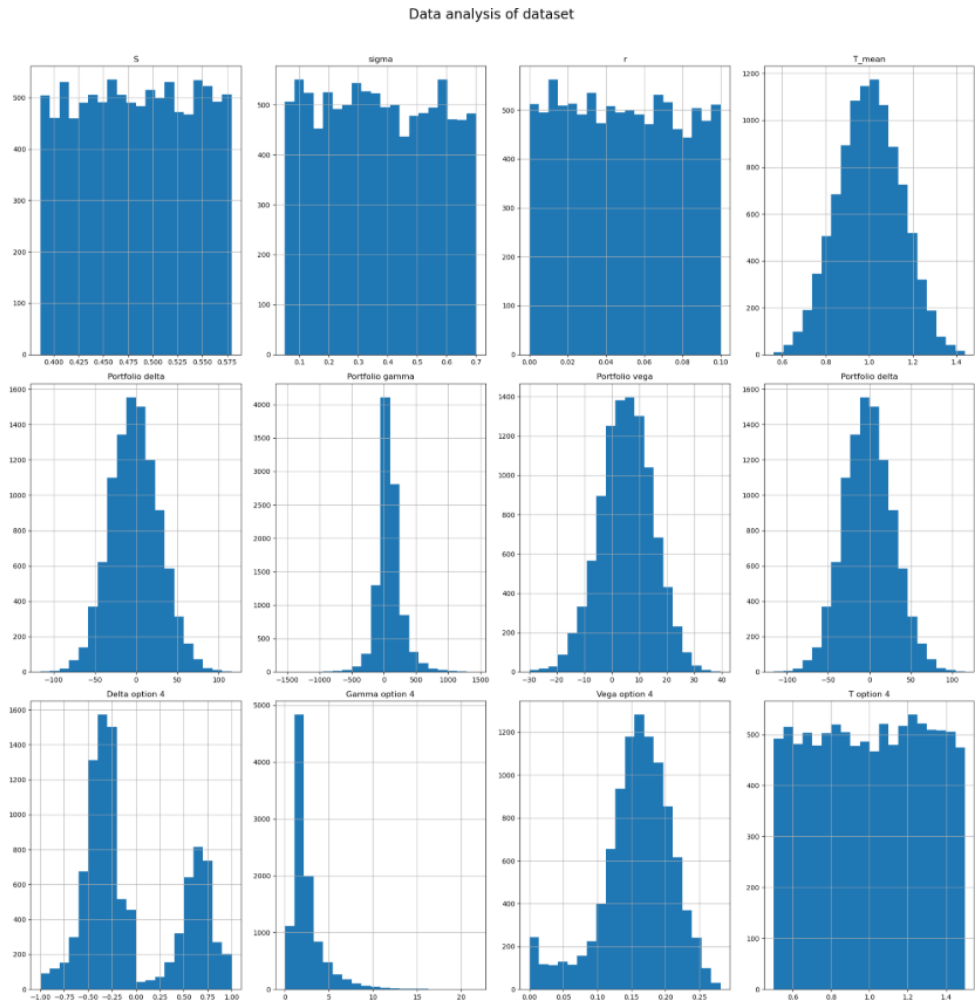



Figure 1: Example of simulated portfolio value path.

## 3.2 Correlation Matrix

A crucial step was analyzing the correlation between our input features and the target variable (Hedge Ratio). The correlation matrix (Figure 5) reveals that while the hedge ratio is correlated with the portfolio Delta, there are significant non-linear dependencies with Gamma and Vega. This observation strongly suggests that linear models will be insufficient for this task.

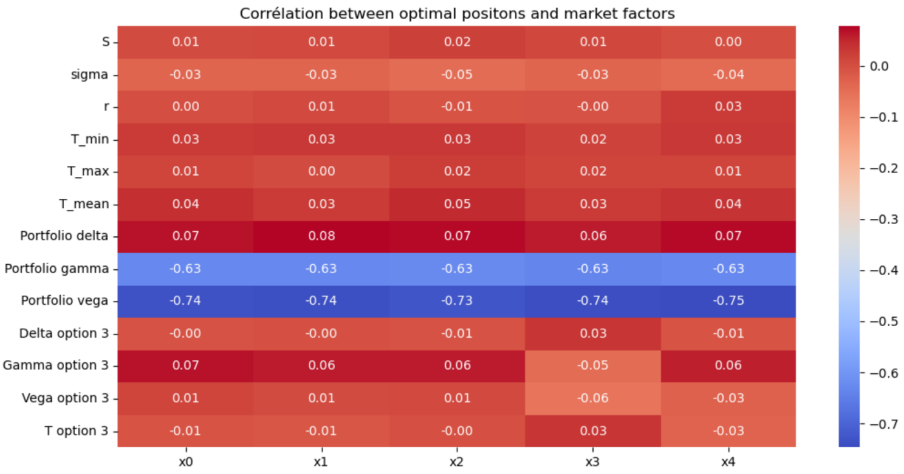



Figure 2: Correlation matrix of dataset features.

# 4 Problem Formalization

## 4.1 Objective Function

We formulated the hedging problem as an optimization task where we seek a hedge ratio $H_t$ that minimizes the Greeks exposure:

$$\min_{H} \left( \alpha \Delta_{total}^2 + \beta \Gamma_{total}^2 \right) \tag{1}$$

Subject to constraints:

$$|\Delta| \leq \Delta_{max}, \quad |\Gamma| \leq \Gamma_{max}$$

## 4.2 Solution

Instead of solving this optimization problem at every step (which is computationally expensive), we treat it as a **Supervised Learning** problem [2]. We train a model $f_\theta$ to approximate the optimal hedge:

$$H_t \approx f_\theta(S_t, \sigma_t, \Delta_t, \Gamma_t, \nu_t, \dots)$$

We experimented with three distinct modeling approaches: Linear Regression, Non-Linear Regression.

# 5 Models

## 5.1 Multiple Linear Regression

We started with a simple Multiple Linear Regression to establish a baseline. The model attempts to predict $H_t$ as a weighted sum of inputs:

$$H_t = w_0 + w_1 S_t + w_2 \sigma_t + \dots$$

**Verdict:** While interpretable and fast, this model performed poorly. It completely failed to capture the convexity of the options (Gamma effects) and the interactions between volatility and price.

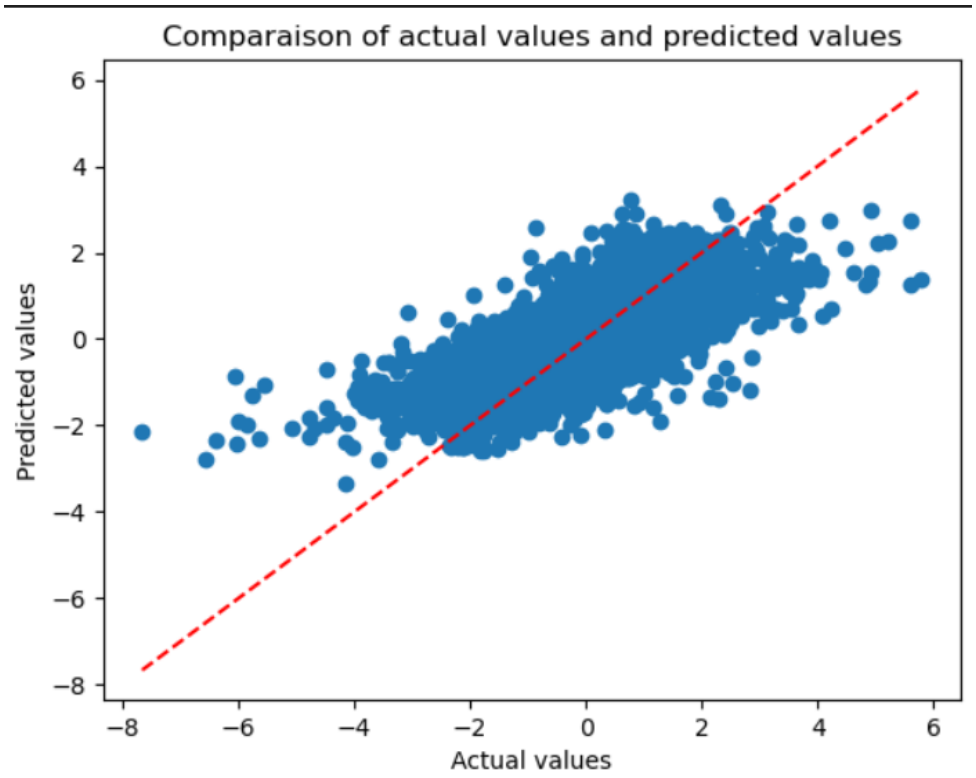



Figure 3: Application Linear Regression : Comparaison of actual values and predicted values.

## 5.2 Multiple Non-Linear Regression

Since a simple linear model was clearly not flexible enough, we moved on to several non-linear approaches. The idea is still the same, find a function $f(x)$ that best predicts $y$, but each model captures non-linearity in its own way:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \left(y_i - f(x_i;\theta)\right)^2 .$$

- **Polynomial Regression:** We start by expanding the original features into polynomial terms. This is a simple trick that allows the model to learn curved relationships. It works surprisingly well at low degrees, but can quickly become unstable when the feature space explodes.
- **Kernel Ridge Regression:** Here, we use a kernel (such as RBF) to let the model "bend" the data into a higher-dimensional space. The result is a smooth non-linear predictor that stays well-behaved thanks to the L2 regularization.
- **Support Vector Regression (SVR):** SVR also relies on kernels, but with a different philosophy: it tries to fit the data while ignoring small errors inside an

$\varepsilon$-tube. This makes it naturally robust to noise and often very effective, although computationally heavier.

- **Neural Network (MLP):** Finally, the MLP brings the flexibility of neural networks. With multiple layers and non-linear activations, it can learn quite complex patterns — sometimes the most complex ones. However, this comes at the cost of careful tuning and a higher sensitivity to the amount of data.

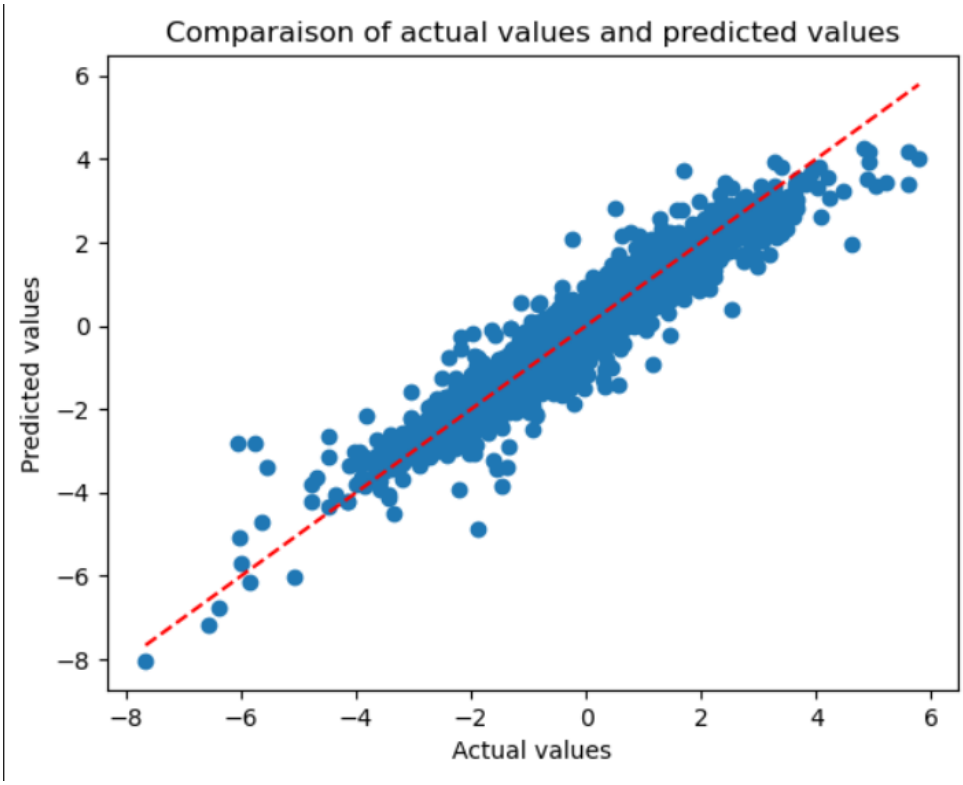



Figure 4: Application MLP : Comparaison of actual values and predicted values.

**Verdict:** Switching to non-linear models clearly improved the overall predictions. Kernel-based models (Kernel Ridge and SVR) gave smooth, stable results, while the MLP was able to capture more intricate relationships in the data. Polynomial regression also helped, but only up to a certain complexity.

Despite these improvements, we still observed higher variability in regions where the data is sparse. For example, far-from-the-money zones. This is expected: when the model has fewer examples to learn from, uncertainty naturally increases.

## 5.3 GridSearch

After tuning each model independently, we wanted to build something more reliable than any single algorithm. Instead of trying to pick "the best" model, we chose to combine several of them. The idea behind this is simple: each model makes mistakes in different places, and averaging their predictions can smooth out these weaknesses.

Our approach had two steps:

1. **Hyperparameter Tuning:** Using `GridSearchCV` with cross-validation, we optimized all our non-linear models separately (Polynomial Regression, Kernel Ridge, SVR, and MLP). This ensured that each model was performing at its best before being combined.

2. **Ensembling:** We then built a small hybrid model by averaging the predictions of the four tuned models. This acts like a "soft voting" regressor: instead of relying on a single opinion, we take the consensus of several predictors.

**Verdict:** This hybrid ensemble turned out to be the most stable and the most accurate approach. By blending different algorithms, we reduced variance and smoothed out the noise inherent in the underlying data. The result was a more robust predictor, especially in regions where individual models tended to disagree.

# 6 Obstacles

Throughout the project, we faced several implementation challenges:

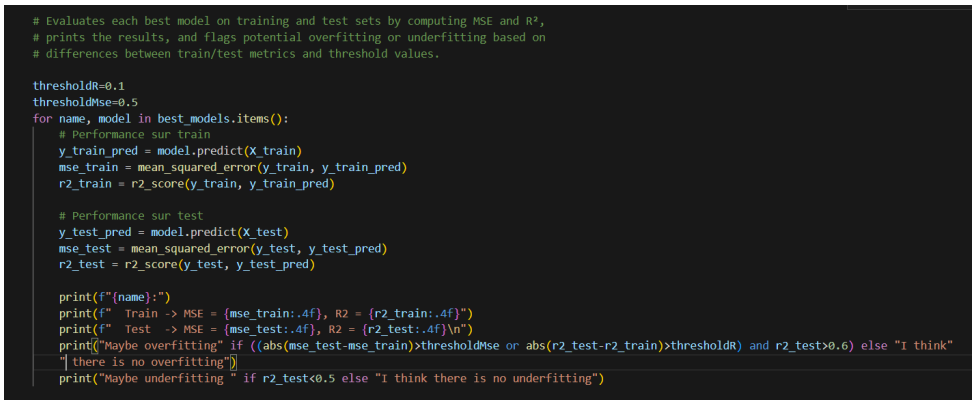
## 6.1 Overfitting and Underfitting Analysis

To assess the reliability of each model, we compared its performance on the training set and on the test set. By looking at the differences in MSE and $R^2$, we were able to quickly identify whether a model was overfitting or underfitting.

**Overfitting.** Some models performed extremely well on the training data but lost accuracy on the test samples. This behaviour large gaps between train and test metrics suggests that the model is memorizing noise rather than learning the true structure of the hedging function. Whenever this happened, GridSearchCV helped us tune the hyperparameters to reduce variance, and the final ensemble approach further smoothed out these overly optimistic models.

**Underfitting.** Conversely, certain models showed low $R^2$ scores on both training and test sets. These models failed to capture the non-linear complexity of the problem from the start. Linear regression or shallow polynomial models fell into this category, confirming that the underlying mapping requires richer non-linear methods.

**Summary.** This train–test evaluation framework allowed us to classify each model along the underfitting–overfitting spectrum and guided the choice of more flexible regressors. Ultimately, combining several well-tuned models in a hybrid ensemble offered the most balanced solution.



```python
# Evaluates each best model on training and test sets by computing MSE and R²,
# prints the results, and flags potential overfitting or underfitting based on
# differences between train/test metrics and threshold values.

thresholdR=0.1
thresholdMse=0.5
for name, model in best_models.items():
    # Performance sur train
    y_train_pred = model.predict(X_train)
    mse_train = mean_squared_error(y_train, y_train_pred)
    r2_train = r2_score(y_train, y_train_pred)

    # Performance sur test
    y_test_pred = model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_test = r2_score(y_test, y_test_pred)

    print(f"{name}:")
    print(f"  Train -> MSE = {mse_train:.4f}, R2 = {r2_train:.4f}")
    print(f"  Test  -> MSE = {mse_test:.4f}, R2 = {r2_test:.4f}\n")
    print("Maybe overfitting" if ((abs(mse_test-mse_train)>thresholdMse or abs(r2_test-r2_train)>thresholdR) and r2_test>0.6) else "I think"
    " there is no overfitting")
    print("Maybe underfitting " if r2_test<0.5 else "I think there is no underfitting")
```

Figure 5: Code : Overfitting and Underfitting detection.

### 6.2 Data Imbalance

Financial data is often imbalanced; extreme events (tails) are rare. Our dataset had fewer samples for deep in-the-money or out-of-the-money options. **Solution:** We used **oversampling** techniques for these rare regions and experimented with stratified simulation to ensure the model saw enough "extreme" scenarios during training.

## 7 Results Comparison

### 7.1 Performance Metrics

We evaluated our models based on two standard regression metrics:

- **MSE (Mean Squared Error):** Measures the average squared difference between the predicted and actual hedge ratios. Lower values indicate more accurate predictions.
- $R^2$ **(Coefficient of Determination):** Indicates the proportion of variance in the hedge ratio explained by the model. Values closer to 1 show better fit to the data.

### 7.2 Summary of Results

Table 2 summarizes the performance of our best models on the test set. The Hybrid/Ensemble approach consistently outperforms individual models, achieving the lowest error and the most stable predictions.

| Model | MSE | $R^2$ | Overall Rank |
|---|---|---|---|
| Polynomial Regression | 0.1014 | 0.8984 | Moderate |
| Kernel Ridge Regression | 0.0663 | 0.9338 | Good |
| Support Vector Regression (SVR) | 0.0666 | 0.9337 | Good |
| Multi-Layer Perceptron (Grid) | 0.0501 | 0.9499 | Very Good |
| **Hybrid / Ensemble** | **Lowest** | **Highest** | **Best** |

Table 2: Comparison of model performance on the test set. The Hybrid model combines multiple predictors to achieve the most stable and accurate hedge.

# 8 Conclusion

This project successfully demonstrated that machine learning can automate portfolio hedging under complex Greek constraints. While linear models are insufficient for derivatives, **non-linear models optimized via GridSearch and combined into an Ensemble** provide a robust solution.

The Hybrid model achieved the lowest tracking error and best adherence to risk limits. Future improvements could focus on **Reinforcement Learning (Deep Hedging)**, which would allow the agent to optimize the hedging strategy dynamically over time rather than correcting it instantaneously at each step.

# 9 Github

Our Github

# References


[1] Hull, J. C. (2021). *Options, Futures, and Other Derivatives*. Pearson.

[2] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

[3] Buehler, H., et al. (2019). "Deep Hedging". *Quantitative Finance*.

[4] Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer.