17. Model risk

Calibration

Calibration process

Calibration process

Calibration is the process of adjusting the parameters of a financial model so that its outputs closely match observed market data. In the context of quantitative finance, calibration is crucial because it ensures that models, such as those used for option pricing or risk management, accurately reflect the current market conditions.

Key Aspects of Calibration

1. **Model Parameters**: Financial models often have parameters that need to be set. For example, in the Black-Scholes model, the volatility of the underlying asset is a parameter that must be determined. In more complex models, such as the Heston model, there are multiple parameters like mean reversion rate, long-term variance, volatility of volatility, and correlation between the asset and its volatility.

2. **Market Data**: Calibration typically uses market data such as prices of liquid instruments (e.g., options, bonds) or historical price data. The idea is to adjust the model parameters until the model's prices for these instruments match the observed market prices as closely as possible.

3. **Objective Function**: During calibration, an objective function is defined to quantify the difference between the model's outputs and the observed market data. A common objective function is the sum of squared errors (SSE) between the model prices and market prices of instruments like options.

4. **Optimization Process**: Calibration involves using optimization techniques to minimize the objective function. This can be done through various numerical methods, such as gradient descent, least squares optimization, or more sophisticated algorithms like genetic algorithms or simulated annealing, depending on the complexity of the model and the nature of the objective function.

5. **Regularization**: Sometimes, to prevent overfitting the model to market data, regularization techniques are employed. This involves adding a penalty term to the objective function, which discourages the model parameters from taking extreme values or from being too sensitive to small changes in market data.

6. **Applications**:

   - **Option Pricing**: Models like Black-Scholes, Heston, or more advanced local volatility and stochastic volatility models need to be calibrated to market data (e.g., implied volatility surfaces) to price options accurately.
   - **Risk Management**: In risk management, calibration ensures that risk models (e.g., Value-at-Risk, CVaR) reflect the actual market risk profile.

> - **Fixed Income**: Calibration is also used in interest rate models, such as the Hull-White or Heath-Jarrow-Morton frameworks, to match the observed yield curves and interest rate derivatives.

Calibrating risk-neutral parameters and real-world parameters are both essential but distinct processes in quantitative finance. Each serves different purposes and uses different data and methodologies.

Risk-Neutral versus historical Calibration

1. **Objective**: Risk-neutral calibration is aimed at accurately pricing derivatives, while real-world calibration is focused on modeling true asset dynamics and understanding risk.
2. **Data Source**: Risk-neutral parameters use current market data (like option prices), while real-world parameters use historical data (like past asset prices).
3. **Measure**: Risk-neutral calibration works under the risk-neutral measure $\mathbb{Q}$, where all securities grow at the risk-free rate. Real-world calibration uses the real-world measure $\mathbb{P}$, which reflects actual investor expectations and risk aversion.
4. **Applications**: Risk-neutral parameters are primarily used for derivative pricing and hedging strategies, whereas real-world parameters are used in portfolio management, risk assessment, and capital allocation.
5. Parameters: in the risk-neutral calibration, the values for the parameters are found that such that the derivatives instruments are matched. Historical calibration can find historical values for those parameters but it can also estimate the drift (remember how in the risk neutral measure, the drift always drops out) and risk premiums.
6. The **Calibration Methodology** also differs. In the risk neutral calibration, the (appropriate) objective function has to be minimized by using and efficient optimization technique where in the historical calibration we can lean on statistical techniques such as Maximulm Likelihood Estimated or Techniques such as regression analysis, neural networks, or clustering. **Scenario Analysis and Stress Testing**: Involves calibrating models to historical extreme events or hypothetical scenarios to assess risk under various conditions.

Challenges in calibration

In risk neutral calibration

- **Non-uniqueness**: Multiple sets of parameters might fit the market data equally well.
- **Model Risk**: Inaccuracies in the model itself can lead to errors in parameter estimation.
- **Changing Market Conditions**: Frequent recalibration may be needed as market conditions change.

In historical calibration

- **Data Quality**: Historical data might be noisy, biased, or not representative of future conditions.
- **Non-Stationarity**: Market conditions change over time, making it difficult to assume constant parameters.

- **Behavioral Factors**: Real-world parameters must account for investor behavior, which can be irrational or unpredictable.

Calibration procedure

For a single asset security, the calibration procedure can be broken down into the following steps

Input

- Obtain market data: this involves option prices at a given time (usually real-time), as well as interest rates, the price of the underlying security. The option prices will come for a variety of strikes and maturities.
- Clean market data: option prices usually come either as the current bid/ask prices or as the last traded price. The last traded price might be stale (old) and the bid-ask might be too large to be useful. It will be apparent that mostly for strikes far away from the market, the quality of the data becomes poor.
- Select the data that will be used to fit the model. Note we might choose to only use a subset of the data.
- Select the model and find the characteristic function.
- Choose initial values for the parameters of the model.

Calibration procedure

- Choose the objective function.
- Choose the optimization algorithm
- Choose the termination conditions for the optimization algorithm.
- Use the FFT method to iterate over parameter sets to
    - calculate the option prices
    - compare the option prices to the market prices
    - calculate the objective function
- Change the parameters to improve the fit.

Output

- An error on the objective function
- A (sub) optimal parameter set.

Optimization

The concepts of **global** and **local minima** and **maxima** are fundamental in mathematics, particularly in calculus and optimization. These terms describe the points at which a function reaches its lowest or highest values within a certain range or over its entire domain.

Optimization is much easier in 1 dimension than for multi-dimensional functions. Note that, other than the Black-Scholes model, all models we discussed have more than 1 parameter and hence the calibration procedure will involve multi-dimensional optimization.

In Gradient descent, Newton method and Nelder–Mead Method we go into detail of the exact optimisation methods.

Global Minima and Maxima

- **Global Minimum**: A point x^* is called a global minimum of a function $f(x)$ if, for all x in the domain of f, the value of $f(x^*)$ is less than or equal to $f(x)$. This means $f(x^*)$ is the lowest value the function attains over its entire domain.

  Mathematically, x^* is a global minimum if:

  $$f(x^*) \leq f(x) \quad \text{for all } x.$$

- **Global Maximum**: Similarly, a point x^* is called a global maximum of a function $f(x)$ if, for all x in the domain of f, the value of $f(x^*)$ is greater than or equal to $f(x)$. This means $f(x^*)$ is the highest value the function attains over its entire domain.

  Mathematically, x^* is a global maximum if:

  $$f(x^*) \geq f(x) \quad \text{for all } x.$$

Local Minima and Maxima

- **Local Minimum**: A point x^* is called a local minimum of a function $f(x)$ if there exists a small neighborhood around x^* such that $f(x^*)$ is less than or equal to $f(x)$ for all x in that neighborhood. In other words, x^* is a local minimum if it is the lowest point in some small region around it, but not necessarily the lowest point over the entire domain of the function.

  Mathematically, x^* is a local minimum if there exists some $\epsilon > 0$ such that:

  $$f(x^*) \leq f(x) \quad \text{for all } x \text{ within the distance } \epsilon \text{ of } x^*.$$

- **Local Maximum**: Similarly, a point x^* is called a local maximum of a function $f(x)$ if there exists a small neighborhood around x^* such that $f(x^*)$ is greater than or equal to $f(x)$ for all x in that neighborhood. In other words, x^* is a local maximum if it is the highest point in some small region around it, but not necessarily the highest point over the entire domain of the function.

  Mathematically, x^* is a local maximum if there exists some $\epsilon > 0$ such that:

  $$f(x^*) \geq f(x) \quad \text{for all } x \text{ within the distance } \epsilon \text{ of } x^*.$$

Visual Representation

- **Global Extremum**: If you think of a function as a landscape, the global minimum would be the lowest valley, and the global maximum would be the highest peak across the entire landscape.
- **Local Extremum**: A local minimum would be like a small dip or a valley that is the lowest point in its immediate area but not necessarily the lowest point in the entire landscape. A local maximum would be like a small hill or peak that is the highest point in its immediate area but not the highest point in the entire landscape.

In practical applications, especially with complex functions such as our objective function, distinguishing between global and local optima is very difficult but important, as optimization algorithms might converge to a local optimum rather than a global one.

Starting values

Many of the optimization methods heavily depend on the starting value of the parameters. It is therefore important to have a good intuition for the involved parameters. One should know for example that volatility of most common currency pairs is around 10%, for equity indices around 20%, for commodities like oil around 40% etc and not use equivalent starting values that lead to a volatility of 1000s or higher. That would make no sense.

There are many ad-hoc methods to choose good starting values. s:

1. **Decomposition and Sequential Calibration**:
   - **Sequential Calibration**: Start by calibrating the model with a subset of parameters, then use the results to inform the calibration of the remaining parameters. This can simplify the process and help ensure more stable results.
   - **Component Models**: Break down the multi-parameter model into simpler component models and calibrate each component separately.
2. **Grid Search and Sensitivity Analysis**:
   - **Grid Search**: Perform a grid search over a range of plausible parameter values. This can provide insights into which parameter ranges are more promising.
   - **Sensitivity Analysis**: Analyze how sensitive the model outputs are to changes in each parameter. This can help you focus on parameters that have a significant impact on the model.
3. **Use of Advanced Optimization Techniques**:
   - **Multi-Objective Optimization**: For models with multiple parameters, consider using multi-objective optimization techniques that can handle trade-offs between different objectives.
   - **Hybrid Optimization**: Combine global optimization methods (e.g., genetic algorithms) with local optimization methods (e.g., gradient-based methods) to efficiently explore the parameter space.
4. **Empirical and Historical Data**:
   - **Parameter Estimation from Similar Models**: Use parameter estimates from similar models or from empirical studies as starting values.
   - **Historical Calibration**: Calibrate the model using historical data to derive reasonable initial estimates for each parameter.
5. **Machine Learning Approaches**:
   - **Machine Learning Models**: Use machine learning techniques to predict reasonable starting values based on historical or simulated data.
   - **Surrogate Models**: Build surrogate models (e.g., neural networks) to approximate the calibration process and suggest starting values.
6. **Regularization Techniques**:
   - **Regularization**: Apply regularization techniques to avoid overfitting and ensure that the parameter estimates are reasonable.
7. **Expert Judgment and Literature**:

- **Expert Input**: Consult domain experts for initial guesses based on their experience and knowledge of the asset or market.
- **Literature Review**: Refer to academic papers and industry reports for typical parameter values used in similar models.

8. **Bootstrap and Monte Carlo Methods**:

   - **Bootstrap**: Use bootstrapped samples of data to estimate parameter ranges and provide initial values.
   - **Monte Carlo Simulation**: Perform Monte Carlo simulations to explore the parameter space and identify reasonable starting values.

9. **Similarity approach**: use prior days parameters or a similar instruments parameters as initial values.

Objective functions

In risk-neutral calibration, the objective function is used to measure how well a model's theoretical prices fit the market prices of options. The goal is to minimize this objective function, thereby finding the best parameter values for the model. Below are some common objective functions, along with their formulas, pros, cons, and intuition:

Selecting the appropriate objective function depends on the specific characteristics of the calibration problem, such as the presence of outliers, the importance of certain data points, and the distribution of errors. Each objective function will lead to a different optimal parameter set and ultimately to different option prices. Especially exotic option prices can be strongly dependent on the parameters.

Least Squares Objective Function

Formula:

$$\text{Objective} = \sum_{i=1}^{n} (\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i})^2$$

Intuition:
This function calculates the sum of the squared differences between the model's prices and the market prices for a set of options.

Pros:

- Simple and easy to compute.
- Works well if the model is a good approximation of the market prices.

Cons:

- Sensitive to outliers and extreme price deviations.
- May not handle large deviations well, potentially skewing the results.

Weighted Least Squares

Formula:

$$\text{Objective} = \sum_{i=1}^{n} w_i(\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i})^2$$

where w_i is the weight assigned to each option.

Intuition:
This function allows for the weighting of different options based on their importance or liquidity. For example, more liquid options might be given higher weights.

Pros:

- Provides flexibility to emphasize more important or reliable data.
- Can help manage the impact of noisy or less reliable prices.

Cons:

- Choosing appropriate weights can be subjective and requires additional assumptions.

Some choices could be to apply weights according to

- open interest in the option
- (inverse) bid-ask spreads: if the spread is larger, we have a wider range of prices that the model can imply. So if such a bid-ask spread is large, we should put less weight on the mid market price.
- Implied volatilities

Absolute Error Objective Function

Formula:

$$\text{Objective} = \sum_{i=1}^{n} |\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i}|$$

Intuition:
This function calculates the sum of the absolute differences between the model's and market prices.

Pros:

- Less sensitive to outliers compared to the least squares approach.
- Provides a more robust measure when prices have extreme deviations.

Cons:

- Less smooth and more difficult to handle analytically compared to squared errors.
- Optimization may be more challenging due to non-differentiability at zero.

Mean Squared Error (MSE)

Formula:

$$\text{Objective} = \frac{1}{n} \sum_{i=1}^{n} (\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i})^2$$

Intuition:
This is the average of the squared differences between the model's and market prices.

Pros:

- Provides an average measure of fit, normalizing for the number of observations.
- Useful for comparing different models.

Cons:

- Similar to least squares, it can be sensitive to outliers.

Mean Absolute Error (MAE)

Formula:

$$\text{Objective} = \frac{1}{n} \sum_{i=1}^{n} |\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i}|$$

Intuition:
This is the average of the absolute differences between the model's and market prices.

Pros:

- More robust to outliers compared to MSE.
- Provides a clear interpretation of average error.

Cons:

- Less smooth, which can make optimization more challenging.
- May not provide as much information on the distribution of errors.

Log-Likelihood Function

Formula:

Assuming normality in errors,

$$\text{Objective} = -\sum_{i=1}^{n} \frac{1}{2} \ln(2\pi\sigma_i^2) - \frac{1}{2} \frac{(\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i})^2}{\sigma_i^2}$$

where σ_i^2 is the variance of the pricing error for each option.

Intuition:
This function assumes that the errors between the model prices and market prices are normally distributed and calculates the log-likelihood of observing the data given the model.

Pros:

- Can handle different error variances across options.
- Provides a probabilistic framework for calibration.

Cons:

- Requires assumptions about the distribution of errors.
- May be complex to compute and interpret.

Chi-Square Objective Function

Formula:

$$\text{Objective} = \sum_{i=1}^{n} \frac{(\text{Price}_{\text{model},i} - \text{Price}_{\text{market},i})^2}{\text{Price}_{\text{market},i}}$$

Intuition:
This function normalizes the squared error by the market price, making it relative to the magnitude of the option price.

Pros:

- Provides a measure of error relative to the size of the market price.
- Can be useful when dealing with options of varying prices.

Cons:

- May not be suitable if market prices are very small or zero.

Implied volatility calibration

Calibrating against implied volatility involves fitting a model to market-implied volatilities rather than directly to option prices. This approach is often used because implied volatilities are typically more stable and less sensitive to market noise than raw option prices. Here's how it works, along with the advantages and disadvantages:

How Calibration Against Implied Volatility Works

1. **Implied Volatility Calculation**:
   - **Market Data**: Collect market prices for a set of options.
   - **Modeling**: Use a pricing model (e.g., Black-Scholes, Heston) to calculate the implied volatility for each option price. This involves solving the inverse of the pricing formula to extract the volatility that would match the market price.
2. **Objective Function**:
   - **Volatility Difference**: Define an objective function that measures the difference between the implied volatilities from the model and the implied volatilities observed in the market.
   - **Example Objective Function**:

     $$\text{Objective} = \sum_{i=1}^{n} (\sigma_{\text{model},i} - \sigma_{\text{market},i})^2$$

     where $\sigma_{\text{model},i}$ is the implied volatility from the model for the i-th option, and $\sigma_{\text{market},i}$ is the market-implied volatility for the same option.
3. **Calibration Process**:
   - **Optimization**: Adjust the model parameters to minimize the objective function. This involves finding parameter values that align the model-implied volatilities with the market-implied volatilities.
   - **Iterative Approach**: Use numerical optimization techniques (e.g., gradient descent, genetic algorithms) to find the best-fit parameters.

Advantages of Calibrating Against Implied Volatility

1. **Stability**:
   - **Reduced Sensitivity**: Implied volatilities are often less noisy than raw prices, as they are derived from market prices and reflect market consensus on future volatility.
   - **Better Fit**: This can lead to a more stable and accurate fit of the model to market expectations.
2. **Arbitrage-Free Prices**:
   - **No Arbitrage**: Implied volatilities are derived from market prices which are assumed to be arbitrage-free. By calibrating to these, the model inherently avoids arbitrage opportunities.
3. **Handling of Smile and Skew**:
   - **Volatility Smile/Skew**: Implied volatilities reflect market phenomena like volatility smiles or skews, which can be critical for accurate modeling, especially in equity markets.

Disadvantages of Calibrating Against Implied Volatility

1. **Complexity**:
   - **Non-Linear Relationships**: The relationship between implied volatilities and model parameters can be complex and non-linear, which can make optimization more challenging.
   - **Numerical Issues**: Numerical stability and convergence issues can arise during the optimization process.
2. **Assumption of Model Validity**:
   - **Model Dependence**: Calibration assumes that the model used to extract implied volatilities is valid. If the model is not a good representation of the market, calibration may be misleading.
3. **Limited Data Points**:
   - **Volatility Surface**: Implied volatilities are available for different strikes and maturities, but there may be fewer data points compared to the raw option prices. This can impact the accuracy of the calibration.

Intuition Behind the Approach

- **Market Implied Volatility**: Implied volatility reflects the market's consensus view on future volatility, which is often a more stable and reliable measure than raw prices, especially when considering pricing models that may not capture all market dynamics.
- **Model Calibration**: By fitting the model to implied volatilities, you align the model with the market's expectations of future volatility rather than raw prices, which can be influenced by additional factors like liquidity or market frictions.

Calibration Instruments

When calibrating a financial model, you can include a variety of instruments to improve the accuracy and robustness of the calibration. Here are some common instruments you might consider:

Vanilla Options

European **Call and Put Options** with various strikes and maturities. Often only out of the money options are considered. Due to the put call parity, using in the money call options or out of the money put options is equivalent. Out of the money options have the advantage of being smaller in price and not distorting the objective function as much.

American options

European Vanilla options have as an advantage that the FFT method can be used to quickly value a wide range of options. However, in some markets only American options are available. One can try and convert the American option prices into European option prices and then use the non-market European options as the input into the calibration. There are a variety of adjustment methods

- Calculate by means of a binomial tree model the implied volatility of the American option and utilize this to price a similar strike/maturity European option.
- Introduce adjustment factors to account for the early exercise premium in American options. For instance, you can modify the European option pricing model to account for the additional value due to the possibility of early exercise. One such example is Merton's adjustment technique to estimate the early exercise premium. This involves adjusting the European option value by an additional term that reflects the value of early exercise.

Option combinations

As the market often trades in option combinations such as Spreads, straddles, strangles, butterflies etc, the calibration can be designed to take those into account straight away.

Exotic options

If market data is available for exotic options, those can be included as well. In particular barrier options or digital options can be relatively liquid in some currency markets. The problem here is that the evaluation of the price of the digital option under the model might be expensive.

Volatility Instruments

It might be beneficial to include **Variance Swaps**, if market prices are available directly into the calibration.

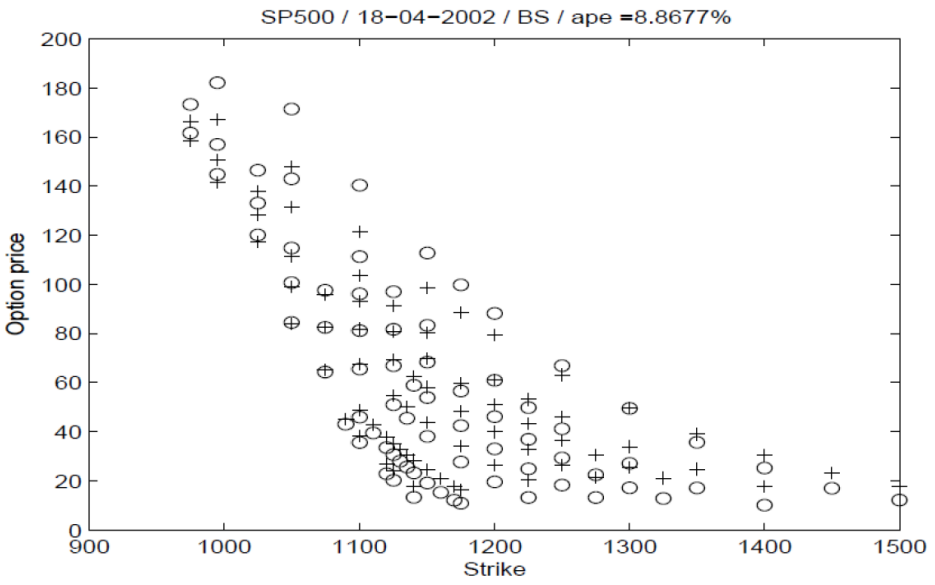
By including multiple instruments, we can greatly improve the accuracy of the calibration. Different instruments can capture various aspects of market behavior and risk, leading to a more robust model. Different types of options and derivatives reflect various market dynamics, such as volatility smiles, skewness, and interest rate changes and including other instruments that also depend on those market dynamics can greatly avoid over-fitting.

Of course, working with a wide range of instruments increases the complexity of the calibration process and the model itself. The model needs to be flexible enough to handle the different characteristics of the instruments included.

Calibration outcome examples

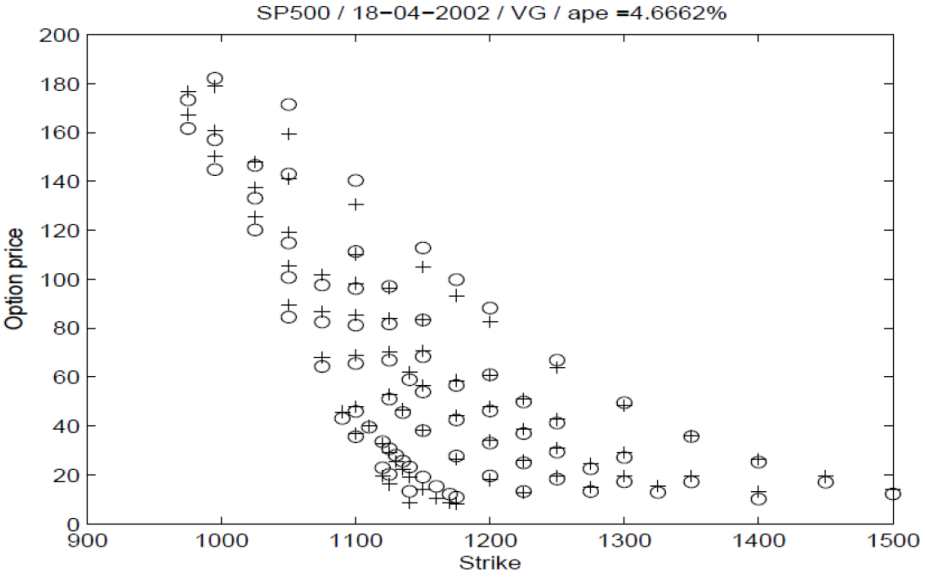
- The first example is the Black-Scholes model. We know that this simple model cannot fit well to the entire volatility surface. We can use the closed form solution as this is the

fastest and most robust calculation method. If we denote the market prices as $+$ and the model prices as o on a chart where on the x-axis we can see the different strikes and on the y axis the option prices, we can observe below how Black-Scholes is a bad fit for almost all market prices.



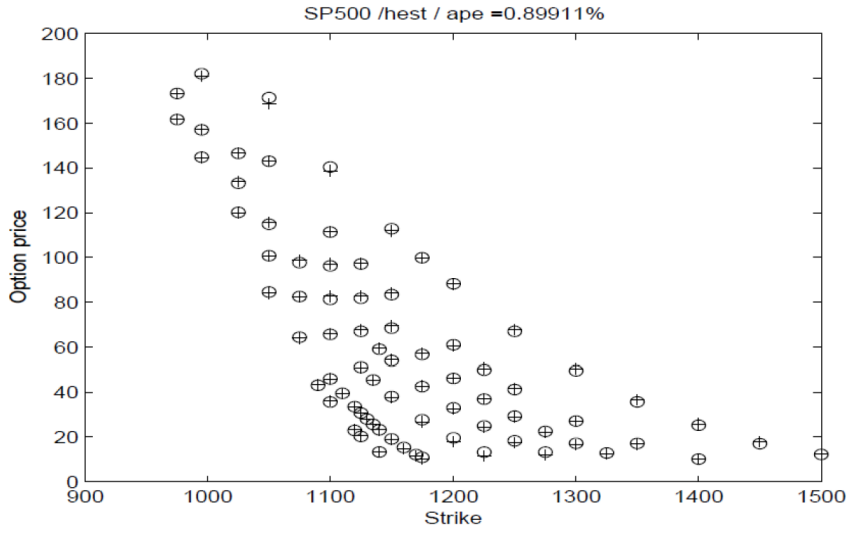
- initial parameters: $(\sigma = 0.1)$
- optimal parameters: $(\sigma = 0.181)$

- The second example is a more flexible model, the Variance Gamma model. This Levy model has 3 parameters and is quite good at capturing skew for a fixed maturity, especially on the short-dated options. This is due to the jump nature of the model that can explain the short dated options (and their deviations from the Black-Scholes prices). For longer dated maturities, this model converges back to the Black-Scholes model so we see similar bad fits on the longer-dated options.



- initial parameters: $(C = 1, \quad G = 5, \quad M = 5)$
- optimal parameters: $(C = 1.3574, \quad G = 5.8703, \quad M = 14.2699)$

- The last example is the Heston model where we can see that it is able to capture most of the volatility surface quite well. It is very typical models that option prices are never exactly matched. The model assumes a certain type of dynamics and although they can be quite flexible in nature (due to the large amount of parameters), there is still a fixed structure underlying the model.



- initial parameters: $(v0 = 0.05, \quad \kappa = 0.5, \quad \eta = 0.05, \quad \lambda = 0.2, \quad \rho = -0.75)$
- optimal parameters: $(v0 = 0.0227, \kappa = 0.2478, \eta = 0.1827, \lambda = 0.3012, \rho = -0.7520)$

Optimization methods

Gradient descent

Gradient Descent is an optimization algorithm used to find the minimum of a function. It is widely used in machine learning and deep learning to minimize the cost function of models during training. The basic idea behind gradient descent is to iteratively adjust the parameters of a function in the direction of the steepest descent (i.e., the negative of the gradient) to reach the function's minimum.

Understanding Gradient Descent

Gradient descent involves the following key steps:

1. **Initialization**: Start with an initial guess for the parameters (weights, biases, etc.).
2. **Compute the Gradient**: Calculate the gradient (partial derivatives) of the function with respect to the parameters. The gradient points in the direction of the steepest increase of the function.
3. **Update the Parameters**: Adjust the parameters in the opposite direction of the gradient to reduce the function's value.
4. **Repeat**: Continue the process until the change in the function's value is below a certain threshold, or after a predefined number of iterations.

Mathematical Formulation

Given a differentiable function $f(\theta)$, where θ represents the parameters to be optimized, the goal of gradient descent is to find the values of θ that minimize $f(\theta)$.

Gradient Descent Algorithm

1. **Initialization**: Start with an initial guess for the parameters $\theta = \theta_0$.
2. **Compute the Gradient**: Calculate the gradient of the function $\nabla f(\theta)$ at the current parameter values θ. The gradient is a vector of partial derivatives:

$$\nabla f(\theta) = \left(\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}, \ldots, \frac{\partial f}{\partial \theta_n} \right)$$

3. **Update the Parameters**: Update the parameters using the formula:

$$\theta = \theta - \eta \nabla f(\theta)$$

   Here, η is the **learning rate**, a positive scalar that controls the size of the steps taken towards the minimum.

4. **Repeat**: Repeat steps 2 and 3 until convergence, which is when the gradient $\nabla f(\theta)$ is close to zero, or the change in $f(\theta)$ is smaller than a predefined threshold.

Practical considerations

There are two practical considerations:

- The choice of η will have an impact on the efficiency (wrong choices can lead to divergences or very slow convergences).
- Sometimes the partial derivatives are not known analytically (or are very hard to calculate), in which case we need to revert to numerical approximations.

Lets explain the latter by means of an example:

Univariate gradient descent

For a univariate function $f(\theta)$ the gradient is simply the derivative $\nabla f(\theta) = f'(\theta)$ which we know can be (Numerical differentiation > Finite Difference Method) approximated numerically as

$$f'(\theta) \approx \frac{f(\theta + h) - f(\theta)}{h}$$

for small values of h.

Example of Gradient Descent

Let's apply gradient descent to a simple quadratic function:

$$f(\theta) = \theta^2$$

Step-by-Step Gradient Descent

1. **Initialization**:

- Let's start with an initial guess $\theta = 4$.

2. **Compute the Gradient**:
   - The derivative of $f(\theta) = \theta^2$ with respect to θ is $f'(\theta) = 2\theta$.
   - At $\theta = 4$, the gradient $f'(4) = 2 \times 4 = 8$.
3. **Update the Parameters**:
   - Choose a learning rate $\eta = 0.1$.
   - Update the parameter θ:

$$\theta = \theta - \eta f'(\theta) = 4 - 0.1 \times 8 = 3.2$$

4. **Repeat**:
   - Compute the gradient at $\theta = 3.2$: $f'(3.2) = 2 \times 3.2 = 6.4$.
   - Update the parameter:

$$\theta = 3.2 - 0.1 \times 6.4 = 2.56$$

   - Continue this process iteratively:

$$\theta_2 = 2.56 - 0.1 \times 5.12 = 2.048$$
$$\theta_3 = 2.048 - 0.1 \times 4.096 = 1.6384$$

   And so on.

As we can see, θ is moving closer to 0, which is the global minimum of the function $f(\theta) = \theta^2$.

We can summarize the convergence in the following table where indeed we see the convergence towards $\theta = 0$ happening.

n	θ_n	$f(\theta_n)$	$f'(\theta)$
1	4.00	16.00	8.00
2	3.20	10.24	6.40
3	2.56	6.55	5.12
4	2.05	4.19	4.10
5	1.64	2.68	3.28
6	1.31	1.72	2.62
7	1.05	1.10	2.10
8	0.84	0.70	1.68
9	0.67	0.45	1.34
10	0.54	0.29	1.07
11	0.43	0.18	0.86
12	0.34	0.12	0.69
13	0.27	0.08	0.55
14	0.22	0.05	0.44
15	0.18	0.03	0.35
16	0.14	0.02	0.28

n	θ_n	$f(\theta_n)$	$f'(\theta)$
17	0.11	0.01	0.23
18	0.09	0.01	0.18
19	0.07	0.01	0.14
20	0.06	0.00	0.12

Reason of convergence

In one dimension, **gradient descent** converges under certain conditions primarily due to the properties of the function being minimized and the choice of the learning rate. Let's explain this in detail using a simple one-dimensional case.

Consider a differentiable function $f(x)$ that we want to minimize. The gradient descent update rule in one dimension is:

$$x_{n+1} = x_n - \eta f'(x_n)$$

where:

- x_n is the current value of the parameter x at iteration n.
- η is the learning rate.
- $f'(x_n)$ is the derivative (gradient) of the function $f(x)$ at x_n.

Heuristic proof

Using Taylor, we know that

$$f(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

if the distance between x_n and x_{n+1} is not too large. So,

$$\begin{aligned} f(x_{n+1}) - f(x_n) = f'(x_n)(x_{n+1} - x_n) = f(x_{n+1}) - f(x_n) \\ = f'(x_n)((x_n - \eta f'(x_n)) - x_n) = -\eta\big(f'(x_n)\big)^2 < 0 \end{aligned}$$

or $f(x_{n+1}) < f(x_n)$.

Proof of convergence

The above heuristic can be more formally proven, which we will now.

Conditions for Convergence

For gradient descent to converge to a minimum, the following conditions are typically required:

1. **Lipschitz Continuity of the Gradient**: The function's gradient should not change too rapidly. Formally, $f'(x)$ is Lipschitz continuous if there exists a constant $L > 0$ such that for all x, y:

   $$|f'(x) - f'(y)| \leq L|x - y|$$

   This means that the gradient is not too steep, which helps prevent large jumps during updates.

2. **Proper Choice of Learning Rate (η)**: The learning rate should be chosen such that $0 < \eta < \frac{2}{L}$. This ensures that the steps taken are neither too large (which can cause overshooting and divergence) nor too small (which can slow convergence).

Why Gradient Descent Converges

To see why gradient descent converges, let's analyze how the function value $f(x)$ changes between iterations:

The update rule is:

$$x_{n+1} = x_n - \eta f'(x_n)$$

The change in x is $\Delta x = x_{n+1} - x_n = -\eta f'(x_n)$.

We want to see how the function $f(x)$ changes with this update:

By Taylor expansion, for small Δx, we can approximate:

$$f(x_{n+1}) \approx f(x_n) + f'(x_n)(x_{n+1} - x_n) + \frac{1}{2} f''(x_n)(x_{n+1} - x_n)^2$$

Substitute $x_{n+1} - x_n = -\eta f'(x_n)$:

$$f(x_{n+1}) \approx f(x_n) - \eta f'(x_n)^2 + \frac{1}{2} f''(x_n)\eta^2 f'(x_n)^2$$

Simplify:

$$f(x_{n+1}) \approx f(x_n) - \eta f'(x_n)^2 \left(1 - \frac{\eta}{2} f''(x_n)\right)$$

Condition for Decrease in Function Value

For convergence, we want $f(x_{n+1}) \leq f(x_n)$. This happens if the term in parentheses is positive:

$$1 - \frac{\eta}{2} f''(x_n) > 0$$

This implies:

$$\eta < \frac{2}{f''(x_n)}$$

If $f(x)$ is convex (i.e., $f''(x) \geq 0$), choosing η small enough (specifically, $0 < \eta < \frac{2}{L}$, where L is the Lipschitz constant for $f'(x)$) guarantees that each step reduces the function's value, leading to convergence.

Example

Consider $f(x) = \frac{1}{2}x^2$, which is a simple quadratic function.

- **Gradient**: $f'(x) = x$
- **Lipschitz Constant**: The gradient's rate of change is constant, $L = 1$.

Gradient descent update:

$$x_{n+1} = x_n - \eta x_n = (1 - \eta)x_n$$

To ensure convergence:

- Choose $0 < \eta < 2$.

If $\eta = 1$, then:

$$x_{n+1} = 0.5x_n$$

This implies that x is halved at each step, and x converges to 0, which is the minimum of $f(x) = \frac{1}{2}x^2$.

Types of Gradient Descent

There are different variants of gradient descent depending on how much data is used to compute the gradient:

1. **Batch Gradient Descent**:
   - **Definition**: Uses the entire dataset to compute the gradient of the cost function.
   - **Pros**: Usually converges smoothly and has a stable path towards the minimum.
   - **Cons**: Can be very slow for large datasets because it needs to compute gradients for the entire dataset at each step.
2. **Stochastic Gradient Descent (SGD)**:
   - **Definition**: Computes the gradient using only one randomly chosen data point at each iteration.
   - **Pros**: Faster per iteration than batch gradient descent and can escape local minima due to its noisy updates.
   - **Cons**: The path towards the minimum is noisier, and it may take longer to converge or may never converge exactly to the minimum.
3. **Mini-Batch Gradient Descent**:
   - **Definition**: A compromise between batch gradient descent and stochastic gradient descent. It computes the gradient using a small, randomly chosen subset (mini-batch) of the data.
   - **Pros**: Offers a balance between the computational efficiency of SGD and the stability of batch gradient descent.
   - **Cons**: The choice of mini-batch size can significantly affect performance and convergence.

Convergence and Learning Rate

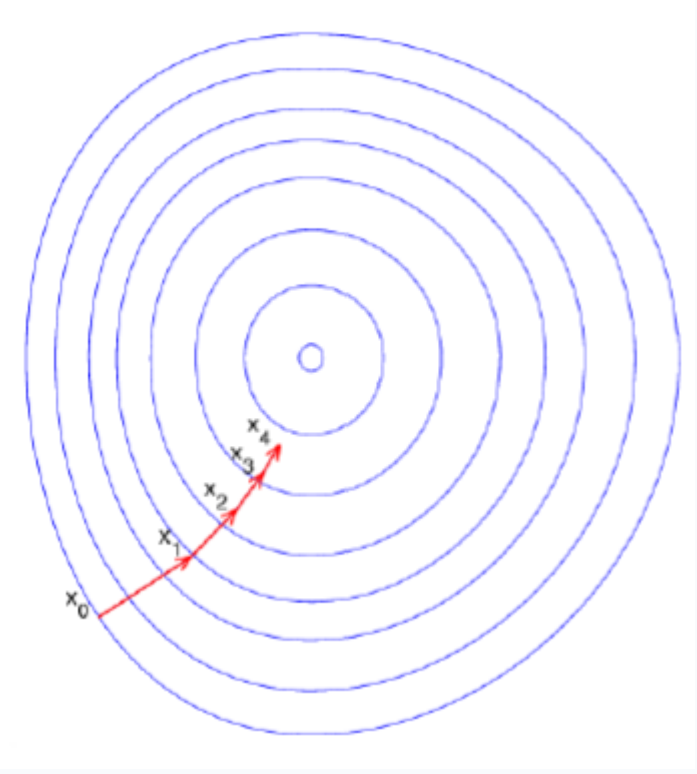
- **Learning Rate (η)**: The learning rate determines the size of the steps the algorithm takes towards the minimum. A too-large learning rate can cause the algorithm to overshoot the minimum and fail to converge. A too-small learning rate can make the algorithm converge very slowly.
- **Convergence**: Gradient descent is said to converge when further updates to the parameters result in negligible changes to the function value. Convergence can be assessed by checking whether the gradient is close to zero or by observing if the change in function value is below a certain threshold.

Challenges with Gradient Descent

1. **Local Minima**: For non-convex functions, gradient descent may get stuck in local minima. However, for convex functions, all local minima are global minima.
2. **Saddle Points**: Points where the gradient is zero but are neither a maximum nor a minimum. These points can slow down convergence since the algorithm might "stall" here.
3. **Vanishing and Exploding Gradients**: In deep learning, especially with deep networks, gradients can become extremely small or large, causing issues in training.
4. **Choosing the Learning Rate**: A proper learning rate is crucial for effective optimization. Adaptive learning rate methods like Adam, RMSprop, and AdaGrad can help by adjusting the learning rate dynamically during training.

Higher dimensions illustration

Assume the function f has a bowl shape. The blue lines in the chart below are the contour lines. This means they are the regions on which the value of the function f is constant. The red arrow originating at a point shows the direction of the negative gradient at that point.



Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see visually in this example how the gradient "descends" to the bottom of the bowl, or to the minimum of the function.

The method cannot be started at a point where the derivative is zero.

Newton method

The **Newton method** is an optimization technique that can be used to choose an optimal step size (learning rate η) in the gradient descent method. Unlike the standard gradient descent,

which updates parameters using only the first derivative (gradient), the Newton method uses both the first and second derivatives (the gradient and the Hessian) to find a more direct path to the function's minimum.

The learning rate η dynamically gets updated by using the second derivative of the function, providing a faster route to the minimum than standard gradient descent, particularly for functions that are well-approximated by quadratics.

Newton's Method for Optimization

The Newton method aims to improve the convergence speed of gradient descent by adjusting the learning rate based on the curvature of the function. It approximates the function locally by a quadratic model and takes steps proportional to the inverse of the second derivative (Hessian in multiple dimensions) to find the optimal learning rate.

Newton's Method Update Rule

Given a twice-differentiable function $f(x)$, Newton's method updates the parameter x as follows:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

where:

- $f'(x_n)$ is the first derivative (gradient) of $f(x)$ at x_n.
- $f''(x_n)$ is the second derivative (Hessian in higher dimensions) of $f(x)$ at x_n.

This formula adjusts the step size based on the curvature of the function, effectively choosing an optimal $\eta = \frac{1}{f''(x_n)}$ dynamically for each iteration.

How Newton's Method Chooses η

In the context of gradient descent, Newton's method provides an optimal step size:

$$\eta_{\text{Newton}} = \frac{1}{f''(x)}$$

Using this in the gradient descent update rule $x_{n+1} = x_n - \eta f'(x_n)$ gives the Newton update:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

This formula ensures that the update considers both the slope (gradient) and the curvature (second derivative) of the function, leading to faster convergence compared to the standard gradient descent.

Example: $f(x) = x^2$

Let's apply Newton's method to the function $f(x) = x^2$.

- **First derivative**: $f'(x) = 2x$
- **Second derivative**: $f''(x) = 2$

Newton's Method Update

Using Newton's update rule:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} = x_n - \frac{2x_n}{2} = x_n - x_n = 0$$

After the first iteration, Newton's method immediately sets $x_{n+1} = 0$, which is the global minimum of $f(x) = x^2$. This demonstrates that for a quadratic function, Newton's method finds the minimum in a single step.

Proof of Newton's Method Convergence

To understand why Newton's method converges faster than gradient descent, let's derive the update rule for a general quadratic function:

Consider a quadratic function of the form:

$$f(x) = ax^2 + bx + c$$

- **First derivative**: $f'(x) = 2ax + b$
- **Second derivative**: $f''(x) = 2a$

The Newton update rule becomes:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} = x_n - \frac{2ax_n + b}{2a} = x_n - \frac{2a(x_n - x^*)}{2a} = x^*$$

where $x^* = -\frac{b}{2a}$ is the location of the minimum. Newton's method directly computes the minimum for quadratic functions in one step.

General Proof of Convergence

For functions beyond simple quadratics, consider the Taylor series expansion of $f(x)$ around x_n:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2}f''(x_n)(x - x_n)^2$$

Newton's method finds the stationary point of this quadratic approximation:

$$0 = f'(x_n) + f''(x_n)(x_{n+1} - x_n)$$

Solving for x_{n+1}:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

By accounting for curvature with $f''(x_n)$, Newton's method moves towards the minimum more directly. If $f(x)$ is convex and smooth, Newton's method converges quadratically, meaning the error decreases exponentially after each step:

$$|x_{n+1} - x^*| = O(|x_n - x^*|^2)$$

This quadratic convergence shows that Newton's method significantly outperforms gradient descent, which typically has linear convergence for convex functions:

$$|x_{n+1} - x^*| = O(|x_n - x^*|)$$

Illustration

Let's apply Newton's method to the function $f(x) = x^4$ to understand how it behaves with a higher-degree polynomial.

Calculate the Derivatives

- **First derivative**: $f'(x) = 4x^3$
- **Second derivative**: $f''(x) = 12x^2$

Newton's Method Update Rule

The Newton's method update rule for finding the minimum is:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

Substitute the derivatives of $f(x) = x^4$:

$$x_{n+1} = x_n - \frac{4x_n^3}{12x_n^2} = x_n - \frac{4}{12}x_n = x_n - \frac{1}{3}x_n = \frac{2}{3}x_n$$

Thus, the update rule simplifies to:

$$x_{n+1} = \frac{2}{3}x_n$$

Convergence Analysis

To see how this update rule behaves, let's analyze its convergence:

- **Iteration 1**: If we start with an initial guess $x_0 = 1$, the first update gives:

$$x_1 = \frac{2}{3}x_0 = \frac{2}{3} \times 1 = \frac{2}{3}$$

- **Iteration 2**: Applying the update rule again:

$$x_2 = \frac{2}{3}x_1 = \frac{2}{3} \times \frac{2}{3} = \frac{4}{9}$$

- **Iteration 3**:

$$x_3 = \frac{2}{3}x_2 = \frac{2}{3} \times \frac{4}{9} = \frac{8}{27}$$

We can see a pattern: the value of x_n is shrinking by a factor of $\frac{2}{3}$ with each iteration.

General Form of Iteration

After n iterations, the value of x_n can be written as:

$$x_n = \left(\frac{2}{3}\right)^n x_0$$

Since $\left(\frac{2}{3}\right)^n \to 0$ as $n \to \infty$, we can conclude that:

$$x_n \to 0 \quad \text{as} \quad n \to \infty$$

This means that Newton's method converges to the minimum of $f(x) = x^4$ at $x = 0$.

Convergence Rate

For $f(x) = x^4$, Newton's method converges **linearly** rather than quadratically. This is because the second derivative $f''(x) = 12x^2$ approaches 0 as x approaches 0, slowing down convergence. In contrast, with a quadratic function $f(x) = x^2$, Newton's method reaches the minimum in one step.

Proof of Convergence Rate

Let's analyze the convergence rate more formally:

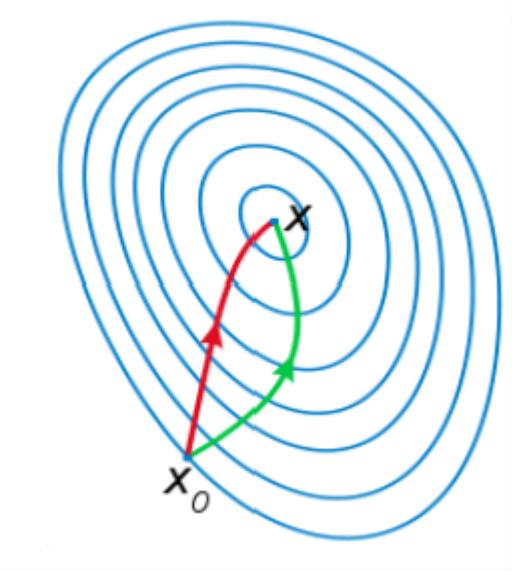
$$|x_{n+1} - x^*| = \frac{2}{3}x_n - 0 = \frac{2}{3}|x_n|$$

This shows that each iteration reduces the distance to the minimum $x^* = 0$ by a constant factor $\frac{2}{3}$. This behavior is indicative of **linear convergence** because:

$$|x_{n+1} - x^*| = O(|x_n - x^*|)$$

Higher dimension illustration

Similar the Gradient descent > Higher dimensions illustration we can demonstrate how the method works. Newton's method also takes into account the curvature and hence leads to a more direct route.



However, it can sometimes lead to zig-zagging and it cannot be started in a point where the derivative is zero.

When the initial value is too far from the true zero, Newton's method may fail to converge.

Nelder–Mead Method

The Nelder-Mead method or downhill simplex method (or amoeba method) is a commonly used non linear optimization technique to minimize a function that does not require the function to be differentiable or smooth. It is often used in situations where derivatives of the function are not available or are expensive to compute, making it suitable for complex, noisy, or non-smooth optimization problems.

The technique was proposed by John Nelder and Roger Mead in 1965 as a technique for minimizing objective functions in high dimensional spaces. Note that the method is only a heuristic, since it can converge to non stationary points.

Key Features of the Nelder–Mead Method

1. **Derivative-Free Optimization**: Unlike gradient-based methods (like gradient descent or Newton's method), the Nelder–Mead method does not require the computation of gradients or second-order derivatives. It relies solely on function evaluations.

2. **Simplex-Based Approach**: The algorithm operates on a geometric figure called a **simplex**, which in n-dimensional space is a shape with $n+1$ vertices. For example, in two dimensions, a simplex is a triangle, and in three dimensions, it is a tetrahedron.

3. **Iterative Process**:

   > - The Nelder–Mead method iteratively adjusts the simplex shape and position in search of the function's minimum. This adjustment is based on comparing the function values at the simplex vertices and applying specific operations to the worst-performing vertex.
   > - In each iteration a new test position gets generated by extrapolating the behaviour of the objective function measured at each test point, arranged as a simplex.
   > - The algorithm chooses to replace one of these test points with a new test point etc

There are many variations of the method, depending on the actual nature of the problem that is being solved.

How the Nelder–Mead Method Works

The Nelder–Mead method uses several key operations to modify the simplex:

1. **Reflection**: Reflect the worst vertex through the centroid of the remaining vertices. This is the most common operation and is used to explore areas that might provide a lower function value.

2. **Expansion**: If reflection finds a better solution, an expansion step is taken further in that direction to accelerate convergence towards the minimum.

3. **Contraction**: If reflection does not improve the solution, the algorithm tries to contract by moving the worst vertex towards the centroid. This helps if the algorithm has overshot the minimum or is on the wrong side of a valley.

4. **Shrink**: If contraction fails to improve, the simplex shrinks towards the best vertex, reducing the step size and refining the search area around the current best-known point.

Algorithm Steps

1. **Initialization**: Start with an initial simplex, which is usually constructed around an initial guess for the minimum.

2. **Iteration**: Perform the following operations iteratively until convergence or a stopping criterion is met:

- **Order**: Evaluate the function at all simplex vertices and order them from the lowest to highest function value: $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1})$
- Calculate x_0 the center of gravity of all points except x_{n+1}
- **Reflection**: Calculate the reflection point $x_r = x_0 + \alpha(x_0 - x_{n+1}$ and evaluate the function there. There are 3 possibilities:
  - **Replace**: $f(x_1) \leq f(x_r) < f(x_n)$: obtain a new simplex point by replacing the worst point x_{n+1} with the reflected point x_r.
  - **Expansion**: $f(x_r) < f(x_1)$: compute the expanded point $x_e = x_0 + r(x_0) - x_{n+1}$.
    - If the expanded point is better than the reflected point, then obtain a new simplex by replacing the worst point x_{n+1} with the expanded point x_e.
    - Else, obtain a new simplex point by replacing the worst point x_{n+1} with the reflected point x_r. Standard values for the procedure are $\alpha = 1$ and $r = 2$.
  - **Contract-Shrink**: $f(x_r) \geq f(x_n)$: compute the contracted point $x_c = x_{n+1} + \rho(x_0 - x_{n+1})$.
    - If the contracted point is better than the worst point, then obtain a new simplex by replacing the worst point x_{n+1} with the contracted point x_c.
    - Else, replace all points, except the best point with $x_i = x_1 + \sigma(x_i - x_1), i = 2, \ldots, n+1$. Standard values are $\alpha = 1, r = 2, \rho = 0.5, \sigma = 0.5$.

3. **Convergence**: The algorithm stops when either
   - the simplex size becomes very small $|x_i - x_{i-1}| < \epsilon_1$, or
   - the function values at the vertices are close enough to each other, indicating that further improvement is minimal: $|f(x_i) - f(x_{i-1})| < \epsilon_2$, or
   - a number of iterations is reached.

Example of Nelder–Mead Method in 2D

Let's consider an example of minimizing a simple function $f(x, y) = x^2 + y^2$ using the Nelder–Mead method in 2D. This function has a global minimum at $(x, y) = (0, 0)$.

1. **Initialization**: Start with a simplex defined by three points, for example, $A = (1, 1)$, $B = (1, 0)$, and $C = (0, 1)$.

2. **Evaluation**: Compute the function values at these points:
   - $f(A) = 1^2 + 1^2 = 2$
   - $f(B) = 1^2 + 0^2 = 1$
   - $f(C) = 0^2 + 1^2 = 1$

   Here, A is the worst point, and B and C are tied for the best.

3. **Reflection**: Compute the centroid M of the best points B and C:

$$M = \left(\frac{1+0}{2}, \frac{0+1}{2}\right) = \left(\frac{1}{2}, \frac{1}{2}\right)$$

Reflect A through M:

$$A_{\text{reflected}} = M + (M - A) = \left(\frac{1}{2}, \frac{1}{2}\right) + \left(\frac{1}{2} - 1, \frac{1}{2} - 1\right) = (0, 0)$$

Evaluate $f(A_{\text{reflected}}) = 0$.

4. **Expansion**: Since $f(A_{\text{reflected}}) = 0$ is better than $f(B) = 1$, try expanding:

$$A_{\text{expanded}} = M + 2(M - A) = \left(\frac{1}{2}, \frac{1}{2}\right) + 2\left(\frac{1}{2} - 1, \frac{1}{2} - 1\right) = (-0.5, -0.5)$$

Evaluate $f(A_{\text{expanded}}) = (-0.5)^2 + (-0.5)^2 = 0.5$.

5. **Contraction and Shrinking**: If needed, perform contraction or shrinking based on subsequent function evaluations.

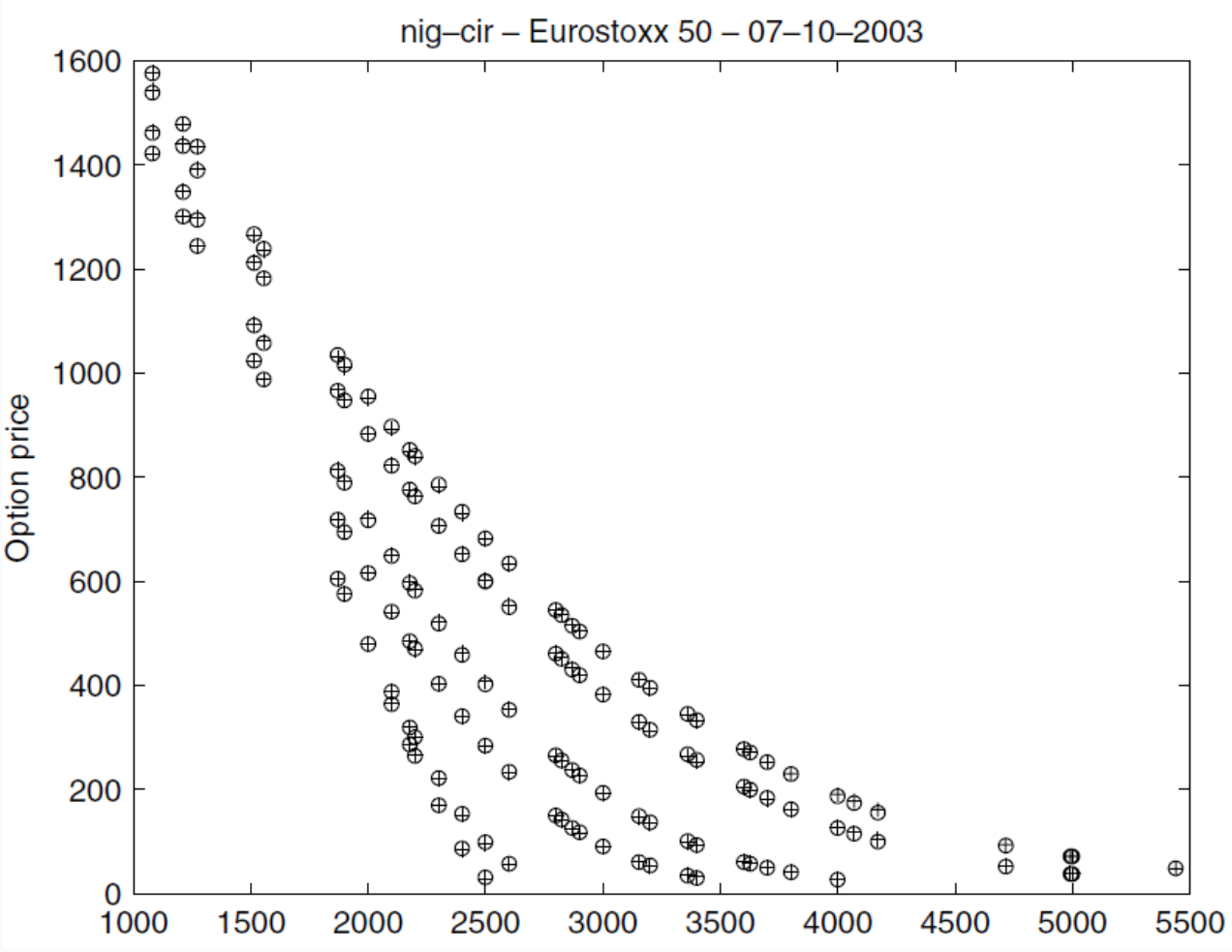
6. **Convergence**: Repeat the steps, refining the simplex until it converges towards the minimum $(0, 0)$.

Perfect calibration, now what

Perfect Calibration, now what

By using flexible models, we can obtain a near-perfect calibration for the volatility surface. We already in Calibration process > Calibration outcome examples that the Heston Stochastic Volatility model is a very good candidate model to achieve this. But there are even more flexible models such as the Bates model that extends Heston to include jumps or some advanced Levy models.

The outcome of the calibration against the option prices is typically excellent for each of those models as in the next chart. Visually it is often hard to tell apart the fits.



We commented that the objective function plays a role as well and we can minimize different objective functions in each of these models and obtain the following errors across models and objective functions.

Model:	rmse	ape	aae	arpe
HEST	3.0281	0.0048	2.4264	0.0174
HESJ	2.8101	0.0045	2.2469	0.0126
BN-S	3.5156	0.0056	2.8194	0.0221
VG-CIR	2.3823	0.0038	1.9337	0.0106
VG-OUΓ	3.4351	0.0056	2.8238	0.0190
NIG-CIR	2.3485	0.0038	1.9194	0.0099
NIG-OUΓ	3.2737	0.0054	2.7385	0.0175

The corresponding optimal parameters can all look very reasonable:

HEST

$\sigma_0^2 = 0.0654,\ \kappa = 0.6067,\ \eta = 0.0707,\ \theta = 0.2928,\ \rho = -0.7571$

HESJ

$\sigma_0^2 = 0.0576,\ \kappa = 0.4963,\ \eta = 0.0650,\ \theta = 0.2286,\ \rho = -0.9900,$
$\mu_j = 0.1791,\ \sigma_j = 0.1346,\ \lambda = 0.1382$

BN-S

$\rho = -4.6750,\ \lambda = 0.5474,\ b = 18.6075,\ a = 0.6069,\ \sigma_0^2 = 0.0433$

VG-CIR

$C = 18.0968,\ G = 20.0276,\ M = 26.3971,\ \kappa = 1.2145,\ \eta = 0.5501,$
$\lambda = 1.7913,\ y_0 = 1$

VG-OUΓ

$C = 6.1610,\ G = 9.6443,\ M = 16.0260,\ \lambda = 1.6790,\ a = 0.3484,$
$b = 0.7664,\ y_0 = 1$

NIG-CIR

$\alpha = 16.1975,\ \beta = -3.1804,\ \delta = 1.0867,\ \kappa = 1.2101,\ \eta = 0.5507,$
$\lambda = 1.7864,\ y_0 = 1$

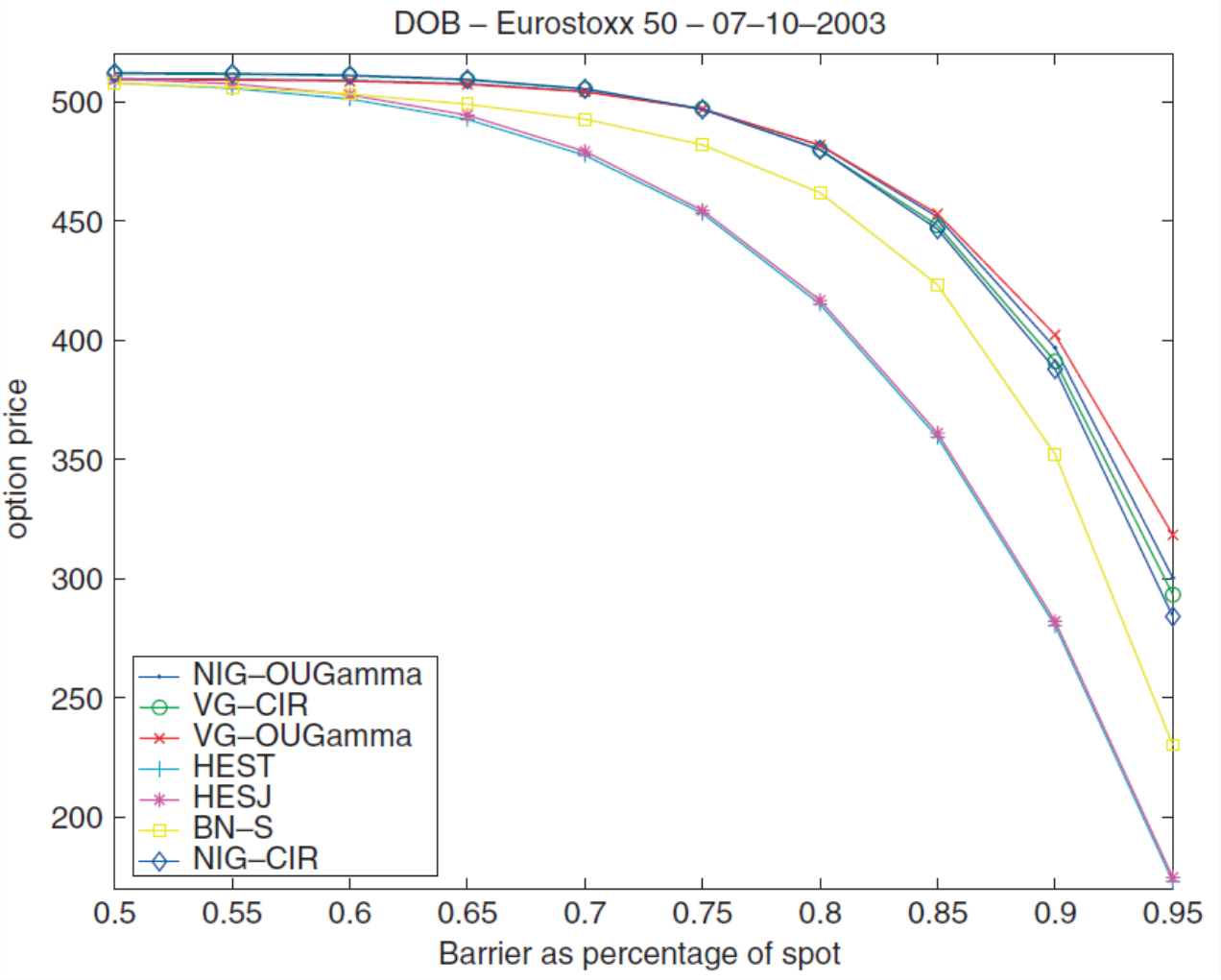
NIG-OUΓ

$\alpha = 8.8914,\ \beta = -3.1634,\ \delta = 0.6728,\ \lambda = 1.7478,\ a = 0.3442,$
$b = 0.7628,\ y_0 = 1$

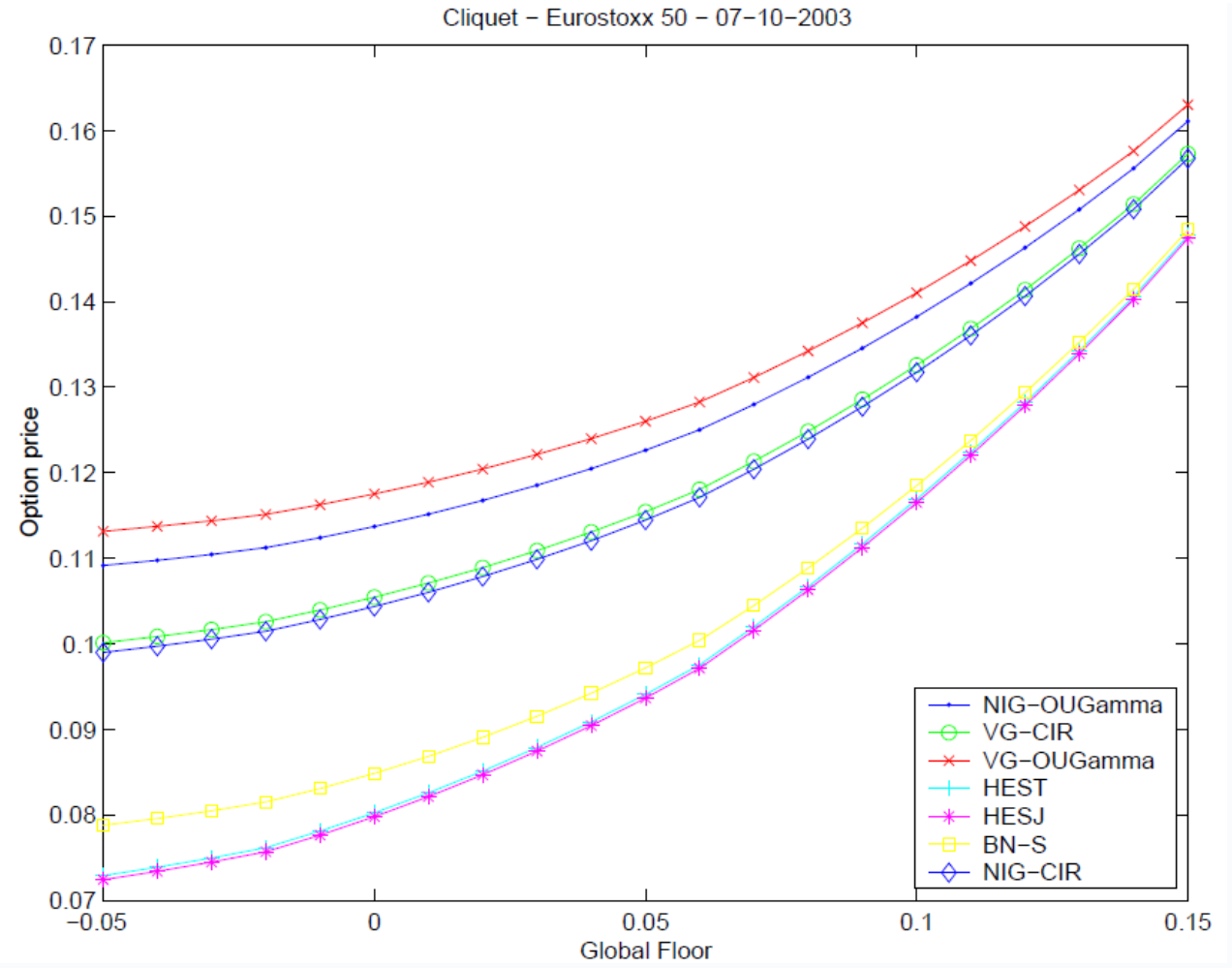
We can then utilize all these models to price a range of exotic options

- Barrier Options Down-and-Out Barrier options (DOB)
- Lookback options
- Cliquet options

One would expect that all these models give a similar qualitative outcome for the price of the exotic option. However, this does not turn out to be the case.



HEST	HESJ	BN-S	VG-CIR	VG-OUΓ	NIG-CIR	NIG-OUΓ
838.48	845.19	771.28	724.80	713.49	730.84	722.34



It is remarkable that, even though the plain vanilla options have been priced equally well across all models, the exotics can be so different. The reason is that the underlying fine-grain properties of the process in the model can have a very important impact on the path-dependent properties of the derivative.

Therefore, it becomes really important to understand the underlying model properties as well as the product properties to match them up well. Furthermore, it also points to the fact that the plain vanilla options surface does not capture all the dynamics of the market. And calibrating to only this source of data, might lead to badly fitted models.