Imperial College London

Imperial College London

Department of Mathematics

Selected Topics in Finance Coursework

Authors:
Muaz Chowdhury (CID: 02291903)
Felix Eychenne (CID: 02299994)
Yanyan Jin (CID: 02297009)

Date: October 19, 2023

1 Introduction

We looked at the paper "Detecting and repairing arbitrage in traded option prices", an arbitrage repair method which formulates the problem as a linear programming problem (LP problem), using model-independent no-arbitrage relations as constraints. The goal of this paper is to make option price data arbitrage-free, by making minimal and only necessary changes. As opposed to, for example, smoothing, which tends to change nearly all of the data. There exist quick and efficient algorithms for solving LP's, which makes this method fast. In this paper, we reproduce some of the results from the original paper, using call option price data from Yahoo Finance. In particular, we implement our own version of their algorithm, apply it to TSLA stock call options, and run some computation time and stress tests.

1.1 Notation

We consider N European call options and m expiries $0 < T_1 < \cdots < T_m$. For each expiry T_i, there are n_i strikes $0 < K_1^i < \cdots < K_{n_i}^i$. We have $N = \sum_{i=1}^m n_i$. Note that here we do not require the grid of prices to be rectangular.

Denote by,

- $\mathcal{T}^e := \{T_i : 1 \le i \le m\}$,the set of expiries.
- $\mathcal{P}^{T,K} := \{(T_i, K_j^i)\}_{1 \le i \le m, 1 \le j \le n_i}$, the set of expiry-strike pairs.
- C_j^i, the time 0 call price of the (i,j)-th call option, which has terminal payoff $(S_{T_i} - K_j^i)^+$.
- $D(T_i)$, the discount factor for the T_i-th expiry.
- $\Gamma(T_i)$, the number of shares owned at time T_i if dividend income is reinvested in shares.
- $F(T_i) = \frac{S_0}{\Gamma(T_i)D(T_i)}$, the forward price at time T_i.

1.2 Static Arbitrage

There are two categories of arbitrage: static arbitrage and dynamic arbitrage. Static arbitrage is exploited by a fixed position in options and the underlying stock at the starting time, while the position of the underlying stock can be finitely modified during future trading. Any other arbitrage is dynamic arbitrage. In contrast, dynamic arbitrage depends on the dynamics and path properties of the assets, hence is dependent on the choice of model. In order to repair the option price data in a model-independent way, we will characterise the constraints of the linear program using only static arbitrage: absence of static arbitrage is a requirement for absence of dynamic arbitrage. So, we need the option price data to be free of static arbitrage at time 0. To apply the method detailed by the paper to a specific model, one would have to add constraints that also characterise the dynamic arbitrage in that model.

The relation between no arbitrage and the existence of an equivalent martingale measure (EMM) is characterised by the First Fundamental Theorem of Asset Pricing (FFTAP) method. Here we use a simplified version of FFTAP, which says that there is no *static-arbitrage* if $\exists \mathbb{Q} \sim \mathbb{P}$ such that,

$$\forall (T,K) \in \mathcal{P}^{T,K} \cup (\mathcal{T}^e \times \{0\}), \quad C_0(T,K) = D(T)\mathbb{E}^{\mathbb{Q}}[C_T(T,K)|\mathcal{F}_0]$$

Therefore, static arbitrage constraints can be characterised just by the relations between the terminal payoff and its discounted value at time 0.

1.3 Constraints of the Call Price Surface

Here we go through the constraints necessary for the call prices to be free of static arbitrage, which will form the linear program we want to solve.

We define, $\forall i, j$, the following normalized quantities:

$$M_{T_i} = \frac{S_{T_i}}{F(T_i)}, \quad k_j^i = \frac{K_j^i}{F(T_i)}, \quad c_j^i = \frac{C_j^i}{D(T_i)F(T_i)} \tag{1}$$

By normalising the quantities in the non-static arbitrage condition from the previous part, we get:

$$c_j^i = \mathbb{E}^{\mathbb{Q}}\left[\left(\frac{S_{T_i}}{F(T_i)} - \frac{K_j^i}{F(T_i)}\right)^+ \big|\mathcal{F}_0\right] = \mathbb{E}^{\mathbb{Q}}\left[\left(M_{T_i} - k_j^i\right)^+ \big|\mathcal{F}_0\right] := c(T_i, k_j^i) \quad \forall i,j$$

where $c(T,k) := \mathbb{E}^{\mathbb{Q}}[(M_T - k)^+|\mathcal{F}_0]$ denotes the normalised time T call price for $T \in \mathbb{R}_{>0}$, $k \in \mathbb{R}_{\geq 0}$.

To construct no arbitrage, we need to have a family of Non-Decreasing in Convex Order (NDCO) marginal measures $\{\mathbb{Q}_T\}_{T\in\mathcal{T}^e} := \left\{\mathbb{Q}(\cdot\big|\mathcal{F}_\mathrm{T})\right\}_{T\in\mathcal{T}^e}$ on $(\mathbb{R},\mathcal{B}(\mathbb{R}))$, with a time-independent mean.

Define a set of functions $s(x,y) : X \times Y \to \mathbb{R}$, where $X, Y \in \mathbb{R}_{\geq 0}$,

$$\begin{aligned}\mathcal{S}(X\times Y) = \{(x,y) \mapsto s(x,y) : \forall x_1 < x_2 \in X, y_1 < y_2 < y_3 \in Y,\\ 0 \leq s \leq 1, s(x_1,\cdot) \leq s(x_2,\cdot),\\ -1 \leq \frac{s(\cdot,y_2) - s(\cdot,y_1)}{y_2 - y_1} \leq \frac{s(\cdot,y_3) - s(\cdot,y_2)}{y_3 - y_2} \leq 0\}.\end{aligned} \tag{2}$$

If $c \in \mathcal{S}(\mathbb{R}_{>0} \times \mathbb{R}_{\geq 0})$, then an arbitrage-free surface $(T,k) \mapsto c(T,k)$ can be constructed. In our case of N option prices, a static arbitrage-free call price surface can be constructed if,

$$\exists c \in \mathcal{S}\left(\mathcal{T}^e \times [0, \max_{i,j} k_j^i]\right) \quad \text{such that} \quad \forall (T_i, k_j^i) \in \mathcal{P}^{T,k}, \quad c(T_i, k_j^i) = c_j^i$$

This condition above can be reformulated into a set of constraints. The paper by Carr and Madan [3] showed that the absence of vertical, calendar and butterfly spread arbitrages are sufficient conditions for no static-arbitrage for a rectangular grid of prices. Since we are considering a more general grid, the classical definitions of vertical, calendar and butterfly spreads have to be extended [2]. To write the constraints in a compact way, as shown in Figure 1, we define the following:

$$\beta(i_1,j_1;i_2,j_2) := \frac{c_{j_1}^{i_1} - c_{j_2}^{i_2}}{k_{j_1}^{i_1} - k_{j_2}^{i_2}} \quad \forall k_{j_1}^{i_1} > k_{j_2}^{i_2} \quad \text{, where} \quad 1 \leq i_1, i_2 \leq m,\ 0 \leq j_1 \leq n_{i_1},\ \text{and } 0 \leq j_2 \leq n_{i_2} \tag{3}$$

A **test spread strategy** is defined as, $\forall 1 \leq i_1 \leq i_2 \leq m$, and $\forall 0 \leq j_1 \leq n_{i_1}, 0 \leq j_2 \leq n_{i_2}$, such that $k_{j_1}^{i_1} \geq k_{j_2}^{i_2}$, by

$$S_{j_1,j_2}^{i_1,i_2} = \begin{cases} -\beta(i_1,j_1;i_2,j_2) & \text{if } k_{j_1}^{i_1} > k_{j_2}^{i_2}, \\ c_{j_2}^{i_2} - c_{j_1}^{i_1} & \text{if } k_{j_1}^{i_1} = k_{j_2}^{i_2}. \end{cases} \tag{4}$$

Define the three types of test spread strategies as follows:
(1) Vertical spread: $VS_{j_1,j_2}^i = S_{j_1,j_2}^{i,i}$ with $k_{j_1}^i > k_{j_2}^i$.
(2) Calendar spread: $CS_{j_1,j_2}^{i_1,i_2} = S_{j_1,j_2}^{i_1,i_2}$ with $k_{j_1}^{i_1} = k_{j_2}^{i_2}$ and $i_1 < i_2$.
(3) Calendar vertical spread: $CVS_{j_1,j_2}^{i_1,i_2} = S_{j_1,j_2}^{i_1,i_2}$ with $k_{j_1}^{i_1} > k_{j_2}^{i_2}$ and $i_1 < i_2$.

A **test butterfly strategy** is defined as, $\forall i, i_1, i_2 \in [1,m]$ such that $i \leq i_1$ and $i \leq i_2, \forall j \in [0,n_i], j_1 \in [0,n_{i_1}], j_2 \in [0,n_{i_2}]$ such that $k_{j_1}^{i_1} < k_j^i < k_{j_2}^{i_2}$,

$$B_{j,j_1,j_2}^{i,i_1,i_2} = -\beta(i,j;i_1,j_1) + \beta(i_2,j_2;i,j)$$

Define the two types of test butterfly strategies as follows:
(1) Vertical butterfly: $VB_{j,j_1,j_2}^i = B_{j,j_1,j_2}^{i,i,i}$.
(2) Calendar butterfly: $CB_{j,j_1,j_2}^{i,i_1,i_2} = B_{j,j_1,j_2}^{i,i,i}$ where i, i_1, i_2 are not all equal.

Figure 1 shows the reduced set of constraints. Note that the authors of the paper use a localistion method for the constraints C6.1 and C6.2, which reduces the the number of constraints from $\mathcal{O}(N^3)$ to $\mathcal{O}(mN^2)$ (the proof can be found in the original paper). All of the constraints are linear inequalities of call prices: we can write these constraints in the form $A\mathbf{c} \geq \mathbf{b}$, where $\mathbf{c} = [c_1^1 \dots c_{n_1}^1 \dots c_{nm}^m]^\top \in \mathbb{R}^N$, R is the number of no-arbitrage constraints, $A = (a_{ij}) \in \mathbb{R}^{R\times N}$, and $\mathbf{b} = (b_j) \in \mathbb{R}^R$ are constant matrix and vector corresponding to coefficients and bounds of the inequalities respectively. If a row of the system $A\mathbf{c} \geq \mathbf{b}$ is not satisfied, then there is an arbitrage. We define ϵ to be the vector of perturbations added to the vector of call prices $\mathbf{c}$, which will give us the arbitrage-free prices, i.e. $A(\mathbf{c} + \boldsymbol{\epsilon}) \geq \mathbf{b}$.

Category	Constraints	Number
C1 Outright	$\forall i \in [1,m],\ c^i_{n_i} \geq 0$	m
C2 Vertical spread	$\forall i \in [1,m],\ j \in [1,n_i]$, $\mathrm{VS}^i_{j,j-1} \geq 0$ and $\mathrm{VS}^i_{1,0} \leq 1$	$N+m$
C3 Vertical butterfly	$\forall i \in [1,m],\ j \in [1,n_i-1],\ \mathrm{VB}^i_{j,j-1,j+1} \geq 0$	$N-m$
C4 Calendar spread	$\forall 1 \leq i_1 < i_2 \leq m,\ j_1 \in [0,n_{i_1}],\ j_2 \in [0,n_{i_2}]$, $\mathrm{CS}^{i_1,i_2}_{j_1,j_2} \geq 0$	$\mathcal{O}(mN)$
C5 Calendar vertical spread	$\forall i^* \in [1,m],\ j^* \in [1,n_{i^*}]$, define $\mathcal{I} := \{i,j : T_i > T_{i^*},\ k^{i^*}_{j^*-1} < k^i_j < k^{i^*}_{j^*}\}$, then $\forall i,j \in \mathcal{I},\ \mathrm{CVS}^{i^*,i}_{j^*,j} \geq 0$	$\mathcal{O}(mN)$
C6.1 Calendar butterfly I (Absolute location convexity)	$\forall i^* \in [1,m],\ j^* \in [1,n_{i^*}-1]$, define $\mathcal{I} := \{i,j : T_i > T_{i^*},\ k^{i^*}_{j^*-1} < k^i_j < k^{i^*}_{j^*}\}$, then $\forall i,j \in \mathcal{I},\ \mathrm{CB}^{i^*,i,i^*}_{j^*,j,j^*+1} \geq 0$;	$\mathcal{O}(m^2N)$
	$\forall i^* \in [1,m],\ j^* \in [2,n_{i^*}]$, define $\mathcal{I} := \{i,j : T_i > T_{i^*},\ k^{i^*}_{j^*-1} < k^i_j < k^{i^*}_{j^*}\}$, then $\forall i,j \in \mathcal{I},\ \mathrm{CB}^{i^*,i^*,i}_{j^*-1,j^*-2,j} \geq 0$;	
	$\forall i^* \in [1,m]$, define $\mathcal{I} := \{i,j : T_i > T_{i^*},\ k^i_j > k^{i^*}_{n_{i^*}}\}$, then $\forall i,j \in \mathcal{I},\ \mathrm{CB}^{i^*,i^*,i}_{n_{i^*},n_{i^*}-1,j} \geq 0$	
C6.2 Calendar butterfly II (Relative location convexity)	$\forall i^* \in [1,m],\ j^* \in [1,n_{i^*}-1]$, define $\mathcal{I}_1 := \{i,j : T_i > T_{i^*},\ k^{i^*}_{j^*-1} < k^i_j < k^{i^*}_{j^*}\}$, $\mathcal{I}_2 := \{i,j : T_i > T_{i^*},\ k^{i^*}_{j^*} < k^i_j < k^{i^*}_{j^*+1}\}$, $\forall i_1,j_1 \in \mathcal{I},\ \forall i_2,j_2 \in \mathcal{I}_2,\ \mathrm{CB}^{i^*,i_1,i_2}_{j^*,j_1,j_2} \geq 0$;	$\mathcal{O}(m^2N)$
	$\forall i^* \in [1,m]$, define $\mathcal{I}_1 := \{i,j : T_i > T_{i^*},\ k^{i^*}_{n_{i^*}-1} < k^i_j < k^{i^*}_{n_{i^*}}\}$, $\mathcal{I}_2 := \{i,j : T_i > T_{i^*},\ k^i_j > k^{i^*}_{n_{i^*}}\}$, $\forall i_1,j_1 \in \mathcal{I},\ \forall i_2,j_2 \in \mathcal{I}_2,\ \mathrm{CB}^{i^*,i_1,i_2}_{n_{i^*},j_1,j_2} \geq 0$	

Figure 1: Reduced set of static arbitrage constraints, from the original paper.

The goal is to find the "minimal" perturbation vector $\boldsymbol{\epsilon}$ such that $A(\mathbf{c}+\boldsymbol{\epsilon}) \geq \mathbf{b}$ is satisfied - the sense of "minimal" will depend an objective function we choose to minimise:

$$\min_{\boldsymbol{\epsilon} \in \mathbb{R}^N} f(\boldsymbol{\epsilon}), \quad \text{subject to } A\boldsymbol{\epsilon} \geq \mathbf{b} - A\mathbf{c} \tag{5}$$

where $f : \mathbb{R}^N \to \mathbb{R}$ is the objective function. In the next part, we will discuss which objective functions are chosen in the original paper.

1.4 Choice Of Objective Function

The authors consider separate objective functions for inclusion and exclusion of the bid and ask prices, i.e. with and without consideration of the liquidity of the options.

In the case of excluding the bid and ask prices, and using the mid-price, the ℓ_1 norm is chosen. This norm is chosen over the ℓ_2 norm and the ℓ_0 norm for various reasons. For example, the ℓ_2 norm generally leads to all prices being affected, while the goal here is to minimise the number of prices that are affected. Using the ℓ_1 norm can make the linear constraints optimisation as an LP problem, allowing us to apply the efficient LP solvers that exist. Using the ℓ_1 norm for the objective function can be written as

$$f(\boldsymbol{\epsilon}) = \sum_{j=1}^{N} |\epsilon_j| \tag{6}$$

In the case of including the bid and ask prices, there might not actually be an arbitrage-free price within the bid-ask bound, so the authors choose not to add this bound as a constraint in the LP: this could result in the optimisation problem not being feasible. Instead, they choose to change the objective function, which will incorporate a penalty for being outside of the bid-ask bound. Now, the goal is to make as many of the prices perturbed by the algorithm fall between their corresponding bid and ask prices, while also trying to make minimal changes to the data. The ℓ_1 norm has the benefit of giving sparse solutions, however this may result in less perturbed prices being within the bid-ask bound. What we want to do here is to choose a new objective function that can lead to a better balance between the two.

Consider the objective function in the form $f(\boldsymbol{\epsilon}) = \sum_{j=1}^{N} f_j(\epsilon_j)$, where $f_j(x) \geq 0 \quad \forall x$. $f_j(\epsilon_j)$ represents the penalty/cost of perturbing the j-th option price by amount ϵ_j. The corresponding derivative will then represent the marginal cost.

Let $\delta_j^a, \delta_j^b > 0$ be the ask-reference and bid-reference spread, respectively. For some specific conditions that the authors want the objective function to satisfy (the details can be found in the original paper), the following function is chosen:

$$f_j(x) = \max(-x - \delta_j^b + \delta_0, \quad -\frac{\delta_0}{\delta_j^b}x, \quad \frac{\delta_0}{\delta_j^a}x, \quad x - \delta_j^a + \delta_0) \tag{7}$$

Then, the objective function considering the bid-ask spread is

$$f(\boldsymbol{\epsilon}) = \sum_{j=1}^{N} f_j(\epsilon_j) \tag{8}$$

$$= \sum_{j=1}^{N} \max(-\epsilon_j - \delta_j^b + \delta_0, \quad -\frac{\delta_0}{\delta_j^b}\epsilon_j, \quad \frac{\delta_0}{\delta_j^a}\epsilon_j, \quad \epsilon_j - \delta_j^a + \delta_0) \tag{9}$$

$$= \sum_{j=1}^{N} \max(-\mathbf{e}_j^\top \epsilon - \delta_j^b + \delta_0, \quad -\frac{\delta_0}{\delta_j^b}\mathbf{e}_j^\top \epsilon, \quad \frac{\delta_0}{\delta_j^a}\mathbf{e}_j^\top \epsilon, \quad \mathbf{e}_j^\top \epsilon - \delta_j^a + \delta_0) \tag{10}$$

where $\mathbf{e}_j$ is the standard basis vector for $\mathbb{R}^N$ with its j-th element being 1 and others being 0, and $\delta_0 \leq \min(\delta_j^a, \delta_j^b)$ represents how costly it is to move prices outside of their corresponding bid-ask bound (smaller δ_0 means it is more costly). Finally, we can write the whole repair problem as the following LP by introducing auxiliary variables $\mathbf{t} = [t_1 \dots t_N]^\top$:

$$\begin{aligned} & \min_{\epsilon, \mathbf{t}} \quad \sum_{j=1}^{N} t_j \\ \text{subject to } & -\epsilon_j - \delta_j^b + \delta_0 \leq t_j, \epsilon_j - \delta_j^a + \delta_0 \leq t_j, \quad \forall j \in [1, N], \\ & -\frac{\delta_0}{\delta_j^b}\epsilon_j \leq t_j, \frac{\delta_0}{\delta_j^a}\epsilon_j \leq t_j, \quad \forall j \in [1, N], \\ & -A\epsilon \leq -\mathbf{b} + A\mathbf{c}. \end{aligned} \tag{11}$$

The result of solving the LP is the optimal perturbation vector $\hat{\boldsymbol{\epsilon}}$, and the corresponding vector of arbitrage-free prices is given by $\hat{\mathbf{c}} = \mathbf{c} + \hat{\boldsymbol{\epsilon}}$. Note that this is the vector of normalised call prices, so to recover the call prices, we can simply multiply each price of the $\hat{\mathbf{c}}$ vector by the corresponding discount factor and forward value: $\hat{C}_j^i = \hat{c_j}^i D(T_i)F(T_i), \quad \forall i, j.$

The authors highlight the results they achieved on FX options data using their implementation of the algorithm, as well as some computation times and stress tests. We have implemented our own version of the algorithm, which we have provided. In the following sections, we summarise our implementation and apply it to options data from Yahoo Finance. We also try to reproduce the computation time and stress tests with our code and compare it with the code written by the authors.

2 Notebook summary

This section will detail our implementation of the repair algorithm using Python. For simplicity, we will consider the TESLA stock in this section since it pays no dividends. Section 5 will detail how the process can be adapted to handle dividend paying stocks or indexes.

We will use the `yahoo_finance` python package to get live data of call prices. We start by importing the available quoted maturities for the TESLA options and transform this data to a day to expiry vector. We then need to choose which strikes to work with: we chose to work with the strikes where we get a quote for each expiry.

$$K = \{k \text{ where } C(T_i, k) \text{ exists } \forall T_i \in \mathcal{T}^e\}$$

However, this vector K can be different and could very well be a two dimensional array with different strikes for each expiry (for example we could work with all the available quotes for each expiry or with an increasing number of strikes with the expiry).

Then, we proceed to import the data in the following way: for each call quote, we extract the ask price and bid price and store them in two dataframes. We define the *reference* price as the mid price and therefore define the reference price dataframe as:

$$\text{reference\_dataframe} = \frac{\text{ask\_dataframe} + \text{bid\_dataframe}}{2}$$

Note that we define the reference price like this to follow the methodology of the authors.

We now move on to the risk-free rate. Since we worked with companies listed on the NASDAQ, it makes sense to work with a proxy for the US risk-free interest rate. We chose to use the OIS rate, the T-bills yields, and the T-bonds yields up to 5 years. We use linear interpolation between these rates to build the deterministic risk-free rate curve:

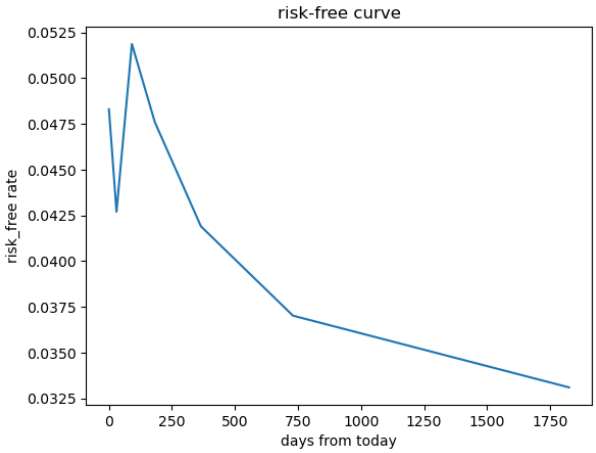


Figure 2: Risk-free curve on the 13th of April

From this curve we are able to infer the discount function

$$D(T) = e^{-\int_0^T r(s)ds}$$

so that the forward price is defined as

$$F(T) = Se^{\int_0^T r(s)ds} = \frac{S}{D(T)}$$

where S is the spot price of the asset (Tesla spot here). Note that we did not take dividends into account here since Tesla does not pay any.

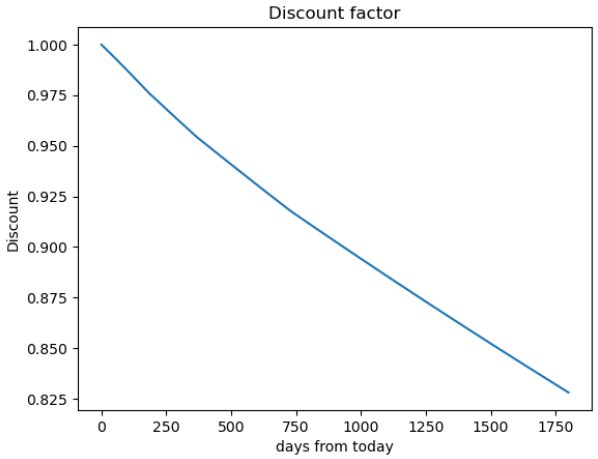


Figure 3: Discount curve on the 13th of April

Next, we normalise the reference price dataframe, dividing every row by the forward function for the corresponding maturity to get the value of c_j^i. We also create the k_j^i dataframe by using the same procedure: the i-th row corresponds to K divided by the forward function for the corresponding maturity.

The following lines of the notebook are the implementation of the `n_violation` function that counts the number of arbitrage in the normalised call surface using the normalised strike dataframe (there is a one-to-one map between every c_j^i and k_j^i). For example, on the 13th of April for the Tesla stock using last price as the reference price the function returns:

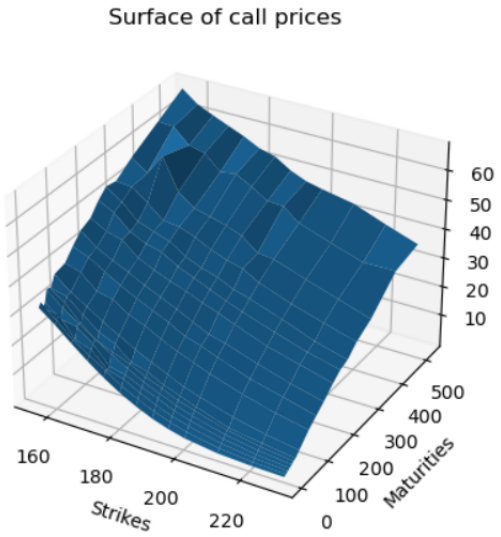
```
n_violation(options_prices_norm,strikes_norm)
```

```
Number of outright violations: 0
Number of vertical spread violations: 16
Number of vertical butterfly violations: 97
Number of calendar spread violations: 0
Number of calendar vertical spread violations: 45
Number of calendar butterfly violations: 483
```

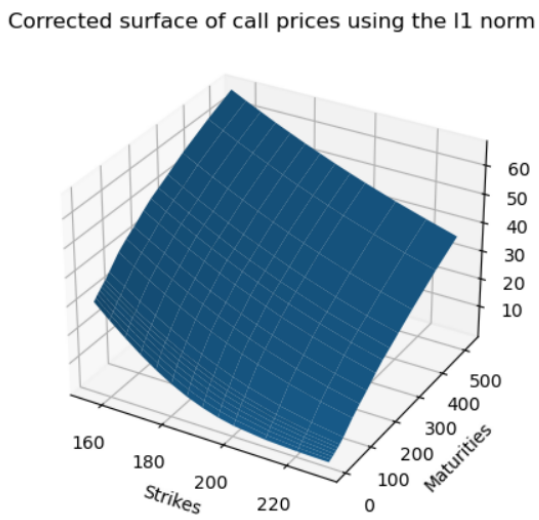
Figure 4: Number of violations for the TESLA "last" call surface on the 13th of April

It is now clear that the goal of the optimization is to reduce the number of arbitrage to zero. The function `n_violation` should therefore prints only zeros once the optimization is complete.

The next cells of the notebook are used to build the matrix A and the vector b that serve the optimization problem. For every constraint 1-6, we add rows to the (initially empty) matrix A and elements to the (initially empty) vector b. Every constraints involves at most 3 call prices, so it is not very hard to do but one need to be extra careful with the indexing. Once done, we can try to solve the optimization problem to find the optimal ε vector of perturbation and add it to the data to make it arbitrage-free. To solve the optimization problem, we used the author's of article [4] found at https://github.com/vicaws/arbitragerepair to make sure we use the same methodology. When the objective function is in l_1 norm, we can pass it as inputs of matrix A, the constraint vector b and the call price vector. However, when the objective function uses l_1-ba norm, we need to pass both the ask-reference and the bid-reference spread. We test the repair algorithm for the TESLA stock on the 13th April using the l_1 norm and use the last price as the reference price (to clearly see the difference in the surface's shape). Finally, we get the following figures:



(a) Tesla before repair



(b) Tesla after repair

```
n_violation(new_prices,strikes_norm)
```

```
Number of outright violations: 0
Number of vertical spread violations: 0
Number of vertical butterfly violations: 0
Number of calendar spread violations: 0
Number of calendar vertical spread violations: 0
Number of calendar butterfly violations: 0
```

(c) Number of violations after repair

Figure 5: Repairing Tesla "last" call surface on the 13th of April with L_1 norm

As expected, Fig.5 shows that the repair method removes arbitrage in the call surface.

The second part of the notebook does exactly the same as the first part, but now it is wrapped as a function. The user only needs to enter a ticker and choose the norm for the repair, and the function returns the optimal vector of perturbation as well with the price dataframe after optimization.

The third part of the notebook deals with stress testing. The function `stress_test` takes an arbitrage-free surface as an input, with λ being the proportion of perturbed prices and σ being the standard deviation in the generated gaussian noise. The function `stats_stress` takes the same arguments as well with a required number of success, and returns the average number of perturbed prices and the proportion of successful optimizations of the algorithm.

3 Stress Testing and Computation Time

In this section, we try to reproduce the results of the stress testing done in the original paper, on our option price data, and then compare the computation times of the repair algorithms. The idea is to start with an arbitrage-free call price surface, add noise to a random proportion $\lambda \in (0,1]$ of the data, and see how well the algorithm can repair the data. We look at what proportion of the data is changed after applying the repair algorithm. Ideally, the proportion of call prices changed by the algorithm will be close to λ.

Let $\mathbf{c} \in \mathbb{R}^N$ denote the set of arbitrage-free call prices. The paper suggests to define the noisy prices in the form,

$$\tilde{c}_j = c_j e^{\xi_i} \quad \text{where } \xi_i \sim \mathcal{N}(0,\sigma) \tag{12}$$

where j represents the indices of the set of random data points that are perturbed, determined by proportion λ.

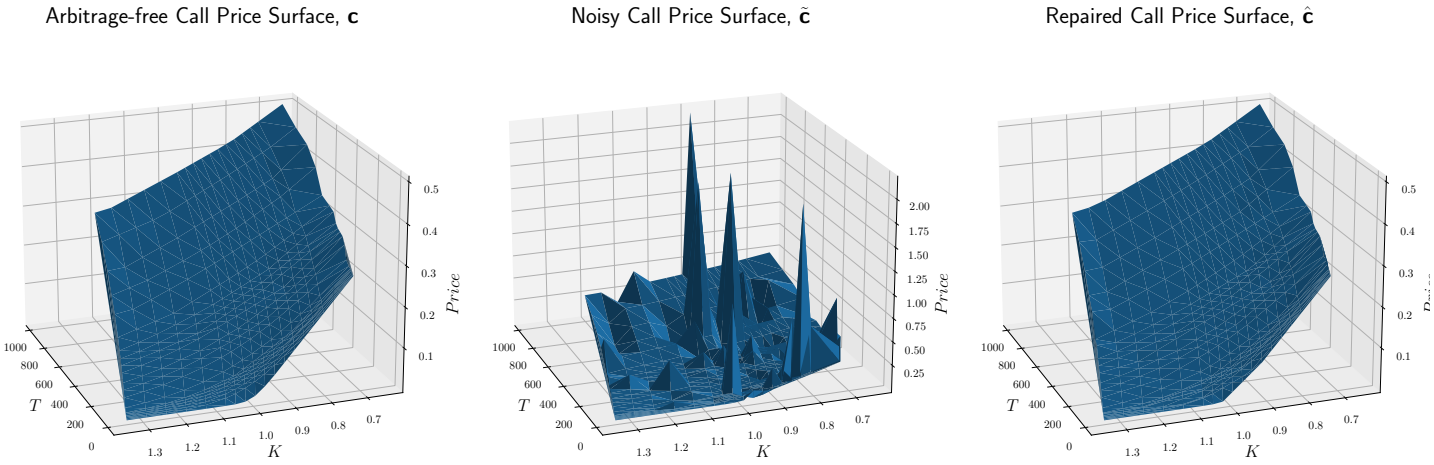


Figure 6: Plots of call surfaces before adding noise, after adding noise, and after repairing. Note that the strike and price axes are normalised here.

Let $\hat{\lambda} := \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}_{\hat{c_j} \neq c_j}$ denote the proportion of data changed by the repair algorithm, where $\hat{\mathbf{c}}$ represents the repaired prices. We simulate the stress test M times and take an average to find $\hat{\lambda}$, and we assess the success of the stress test by $\hat{\lambda} - \lambda$. The original paper omitted the number of times that the algorithm failed to repair the noisy call price surface, but we have added that in our test results. If the algorithm was unable to find a solution, a new set of noise was generated and we continued the process. The results in table 1 show the results of the stress testing both on the code written by the authors and on our code.

It should be noted that when stress testing the code written by the authors, we found that after adding noise to the call price surface and then repairing it, the resulting call price surface in most cases still contained a bit of arbitrage, i.e the algorithm was not completely removing all arbitrage. Assuming this is due to a bug in their code, this may explain why $\hat{\lambda} - \lambda$ and the fail rate of our code was significantly higher than theirs - this is because the necessary conditions for their algorithm to consider the repair to be a "success" are more relaxed, hence less call prices need to be perturbed.

With our code in particular, it's clear that as the proportion of arbitrage-free prices that are perturbed increases, the number of failed attempts of the algorithm to find a solution increases quickly, reaching a failure rate of around 70% with $\lambda = 0.5$. Of course, this level of noise is extremely high (the prices used are normalised so they take a value between 0 and 1) and hence quite unrealistic.

For the computation time, we tested the speed of the repair algorithm on data from Yahoo Finance, for different sizes of option data sets. The results are shown in Table 2. Our implementation of course was not heavily optimised, as this was not the main focus of our report, hence we see that the code of the authors is much faster. However, the way the computation time scales in increased number of data points seems to be similar in both cases: we see roughly that as the number of data points double, the computation time quadruples. Note that these are the computation times of repairing the initial call option price data taken from Yahoo Finance. The

Noise $\sim \mathcal{N}(0,1)$, $M = 50$		ℓ^1-norm		$\ell^1 - BA$-norm	
		Our Code	Authors Code	Our Code	Authors Code
$\lambda = 0.1$	$\hat{\lambda} - \lambda$	29.64%	13.18%	34.95%	35.13%
	Fail rate	7.41%	1.00%	7.41%	10.7%
$\lambda = 0.25$	$\hat{\lambda} - \lambda$	26.93%	19.26%	51.24%	51.24%
	Fail rate	48.02%	3.85%	37.5%	53.7%
$\lambda = 0.5$	$\hat{\lambda} - \lambda$	22.54%	23.24%	43.11%	42.4%
	Fail rate	68.36%	28.57%	59.02%	75.73%

Table 1: Stress test results for different values of λ on the TSLA data on the 15th April

	ℓ^1-norm		ℓ^1-BA norm	
Number of option prices, N	Our Code	Authors Code	Our Code	Authors Code
$N = 72$	$1.34s$	$0.089s$	$1.23s$	$0.081s$
$N = 162$	$4.7s$	$0.31s$	$4.53s$	$0.31s$
$N = 231$	10.1	$1.07s$	$10.1s$	$0.90s$
$N = 340$	$18.8s$	$1.68s$	$18.4s$	$1.67s$

Table 2: Computation time of repairing TSLA data directly from Yahoo Finance on the 15th April for different values of λ

computation time of repairing the noisy price data in the stress testing took significantly longer - for some idea of the time taken, with $\lambda = 0.5$, a simple test of 7 runs using the "timeit" function showed that it took $193s \pm 222s$ to get one successful iteration.

4 Dividends handling and criticisms

This section will detail how dividends can be handled when repairing the surface by suggesting another way of doing things that is natural to the construction of complex tools such as exotic options pricers.

Let τ be the time of payment of a dividend of value α. Let τ^- be the time before the dividend is paid and τ^+ the time just after. We get the following relationship:

$$S_{\tau^+} = S_{\tau^-} - \alpha \tag{13}$$

This also means that for two calls on the stock option, one before τ and one after we get the following relation:

$$\begin{aligned} C(\tau^-, K+\alpha) &= P(\tau^-)\mathbb{E}[(S_{\tau^-} - K - \alpha)^+] \\ &= P(\tau^+)\mathbb{E}[(S_{\tau^-} - K - \alpha)^+] \\ &= P(\tau^+)\mathbb{E}[(S_{\tau^+} - K)^+] \\ &= C(\tau^+, K) \end{aligned} \tag{14}$$

This means that we get jumps in the call surface that are intended, and we need to take those into account when repairing the surface.

Let $t_1 < ... < t_n$ be the dates corresponding to dividend payments for a given stock with European-style options and let $\alpha_1, ..., \alpha_n$ be the corresponding estimated cash payments at those dates. One can get this information with data providing companies such as Bloomberg and their BDVD function, or other models. Following the authors procedure, we define Γ(T) as the number of shares which will be owned by time T if dividend income is invested in shares and the new forward (sufficiently liquid to neglect bid-ask spread) price $F(T) = \frac{S}{\Gamma(T)D(T)}$. Therefore, one can compute $F(T_i) \quad \forall T_i \in \mathcal{T}^e$, and follow the usual procedure depicted in the parts above, then switch back to the repaired set of prices. However, repairing the call price surface is often the starting point for more complex procedures such as using those prices to build an arbitrage-free implied volatility surface that one wants to parameterize in order to fit a (stochastic) local volatility model to the data. Therefore, it would be of use to handle dividends once and for all: the author's methodology to handle dividends does not specify clear dynamics for the stock's price. Following Mr Buehler's article [1] and writing (for simplicity, we do not consider credit risk):

$$\boxed{S_t = (F_t - D_t)X_t + D_t} \tag{15}$$

where

$$D_t = \sum_{\tau > t} P(t, \tau_j)\alpha_j \tag{16}$$

(in practice we stop at the last estimation of dividends α_n), and

$$F(T) = F(0,T) = e^{\int_0^T r(s)ds} S_0 - \sum_{0 < \tau_j \leq T} P(\tau_j, T)\alpha_j, \quad \text{with} \quad P(\tau_j, T) = e^{-\int_{\tau_j}^T r(s)ds} \tag{17}$$

Essentially, everything in (15) is deterministic except for the process X_t which we will deduce from S_t.

To do so, Mr. Buehler writes for a given strike K and expiry T:

$$\begin{aligned} C(K,T) &= P_T\mathbb{E}((S_T - K)^+) \\ &= P_T\mathbb{E}(((F_T - D_T)X_T + D_T - K)^+) \\ &= P_T\mathbb{E}(((F_T - D_T)X_T + D_T - (F_T - D_t)x - D_T)^+) \quad \text{with } x = \frac{K - D_T}{F_T - D_T} \\ &= P_T(F_T - D_T)\mathbb{E}((X_T - x)^+) \\ &= P_T(F_T - D_T)\mathbb{C}(x,T) \end{aligned} \tag{18}$$

This allows to deduce new call prices $\mathbb{C}(x,T)$ and strikes x for the process X_t in the same fashion as in [4]: for two given 2d-dimensional array of strikes and call prices on S_t, we deduce the two new 2d-arrays corresponding to the call prices and strikes on X_t.

Mr. Buehler showed that X_t can be seen as a "clean" stock price with $X_0 = 1$ that pays no dividends in a world

with deterministic risk-free interest rates always equal to $r_X(T) = 0 \; \forall T$ which in turn yields $F_X(T) = 1 \; \forall T$. This is interesting here because it means the optimization problem can be written and solved for the X_t's call surface directly without further transformation, and one can use the resultant surface to, for example, build an implied volatility surface and then use Dupire's formula to deduce the local volatility surface on the process X_t. Indeed, Mr. Buehler showed that the stock price follows the local volatility dynamics

$$dS_t = S_t(r(t)dt + \sigma_{loc}(S_t, t)dW_t) - \sum_j \alpha_j \mathbb{1}_{t=\tau_j} \tag{19}$$

and that there is a one-to-one map between the σ_{loc} function in (19) and the local volatility function on the X_t process.

With this setup, the repairing of the call price surface can be seen as a natural step that fits the process of building exotic options pricers and can help getting a more robust calibration to the data. Furthermore, it does not require any other data than the sole dividend estimate for the stock.

5 Discussions

Very often, the mathematical models used in the markets require data to be calibrated. Bloomberg, for example, allows investors to export live or historical market data, but this data can sometimes present anomalies for calibration that can be explained by intense market conditions, low liquidity, etc. It is therefore essential to pre-process this data before using it for more complex model calibration, as data with too many or too extreme anomalies can cause the calibration (of a parametric model for example) to fail and have unfortunate consequences for market participants.

The paper we have studied here proposes a simple, fast, robust and flexible method to pre-process the price surface of calls on a stock or an index, eliminating arbitrage among the points of the surface. Starting from a simple assumption (stock not paying dividends and subject to European options), it is possible to extend this work to more complex cases such as stocks/indexes paying dividends and subject to American style options. This can be done by:
- Using the authors' method for dividend management or the one developed in part 4.
- De-Americanising the options to obtain a price surface for European calls.
It is also possible to take into account the credit risk by modifying the forward function with a new intensity corresponding to the risk.

The method developed by the authors of [4] is fast but the number of constraints and thus the complexity increases quickly with the number of calls on the surface (in particular quadratically with the number of maturities). The method is fairly robust: the GLPK solver has always found the optimal perturbation vector in the case of non-dividend paying stocks. On the other hand, our stress tests have shown that when the surface becomes too perturbed the algorithm has more difficulties to converge and sometimes fails, which is not mentioned by the authors. This authors have also only detailed removing static arbitrage: removing dynamic arbitrage for a specific model will, in general, lead to much more constraints. This will of course increase computation time, and could potentially even result in an infeasible optimisation problem.

The intensity of the perturbations to make the surface free of arbitrage is stronger for the shortest maturities which is a good sign: indeed options with (very) short maturities are no longer liquid and this leads mathematically to more arbitrage. The prices of calls with medium and long maturities are not disturbed as much, which avoids introducing a bias for possible subsequent calibrations. Finally, the methodology used is simple to understand and implement.

In conclusion, this methodology provides an elegant and efficient answer to an essential and often overlooked problem, and the implementation cost is cheap. However, it might be necessary to determine precisely when the algorithm fails and to find another way of doing it when this happens (by performing several sub-calibrations in parallel, by adding noise, by parsimoniously adding optimal points, by coupling the method with other existing methods etc).

References

[1] Hans Buehler. Volatility modelling with cash dividends and simple credit risk. pages 10

[2] L. Cousot and M. Street. Necessary and sufficient conditions for no static arbitrage among european calls. 2004. pages 2

[3] D. B. Madan. P. Carr. A note on sufficient conditions for no arbitrage. pages 2

[4] Sheng Wang Samuel N. Cohen, Christoph Reisinger. Detecting and repairing arbitrage in traded option prices. pages 6, 10, 12