1-31-2021

Forecasting Optimal Parameters of the Broken Wing Butterfly Option Strategy Using Differential Evolution

David Munoz Constantine
Portland State University

Recommended Citation

Munoz Constantine, David, "Forecasting Optimal Parameters of the Broken Wing Butterfly Option Strategy Using Differential Evolution" (2021). *Dissertations and Theses.* Paper 5643.
https://doi.org/10.15760/etd.7515

Forecasting Optimal Parameters of the Broken Wing Butterfly Option Strategy using Differential Evolution

by

David Munoz Constantine

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

Thesis Committee:
Richard Tymerski, Chair
Garrison Greenwood
Bart Massey
Fei Xie

Portland State University
2020

Abstract

Obtaining an edge in financial markets has been the objective of many hedge funds, investors, and market participants. Even with today's abundance of data and computing power, few individuals achieve a consistent edge over an extended time. To obtain this edge, investors usually use options strategies. The Broken Wing Butterfly (BWB) is an options strategy that has increased in popularity among traders. Profit is generated primarily by exploiting option value time decay. In this thesis, the selection of entry and exit BWB parameters, such as profit and loss targets, are optimized for an in-sample period. Afterward, they are used to assess profitability during an out-of-sample period. The optimization takes place for over a decade of historical options data of the S&P exchange-traded fund (symbol: SPY). The importance of selecting an optimal strike mapping method is emphasized. Of the three mapping methods considered, the normalized strike mapping method was found to be superior. The optimization of the parameters was performed with a differential evolution (DE) evolutionary algorithm. The objective function to optimize took into consideration the strategy's cumulative profits and maximum equity drawdown. The out-of-sample trades' performance shows that information from past trades can be used to trade in the future successfully.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Background

1.1 Introduction

From the moment that mathematician and hedge fund manager, Edward Thorp, pioneered mathematical models to beat the stock market, investors have increasingly relied on mathematical and quantitative models to obtain an edge in the market. These types of traders are known as "quants" (quantitative traders). They took advantage of faster computers and an abundance of data to create new trading models.

Traders can choose over many financial instruments and derivatives to bet on the market. An instrument that has increased in popularity are option contracts. They enable a trader to profit by predicting if stock prices are going up or down, but also by predicting if prices will not move, or if there are changes in volatility, time, and more. Typically, a trader purchases multiple options simultaneously, and these combinations are known as options strategies. When using a strategy, a trader has many parameters to consider, such as expiration dates, strike prices, when to enter, when to exit, and more.

The Broken Wing Butterfly (BWB) is an options strategy that has increased in popularity among options traders. This strategy can be used as an income generator, and some prominent traders ([1], [2], and [3]) have demonstrated that the BWB generates consistent profits while keeping active management to a minimum. The BWB may be entered into, and later exited, when favorable or unfavorable conditions appear, resulting in either a profit or a loss for the trade. In this thesis we aim to find

an optimal set of parameters for the BWB strategy over a defined period, and evaluate if these optimal parameters are generalizable for future trades. Essentially, it aims to evaluate if past information from the stock market contains valuable information to trade in the future.

Due to the exponential search space of the strategy's parameters, a complete search of all possible values is not feasible. Traditional optimization methods that rely on gradients, such as stochastic gradient descent or newton's method, are impractical for this problem due to it being non-differentiable. Therefore, in this thesis we used differential evolution. It has been shown to have more outstanding performance than other non-differentiable optimization methods when the values to optimize are continuous [4].

The function to optimize (called the fitness function) was crafted to maximize risk-adjusted returns. It achieves this by maximizing profits while keeping equity drawdown to a minimum. The BWB strategy was optimized over a set period (the in-sample period), and the resulting optimal parameters were used for trading in the future (the out-of-sample period).

1.2 Financial Options Fundamentals

Options are financial instruments that enable market participants to speculate or hedge positions. The largest public options exchange is the Chicago Board of Options Exchange (CBOE). There exists two types of options, *calls* and *puts*, and two types of option styles, American and European. American call options give the buyer the right (but not the obligation) to buy the underlying asset for a specific price (known as the strike price) by a specific expiration date. In contrast, the call option seller has the obligation to sell the main asset for the same conditions if forced by the buyer. American put options give the buyer the right to sell the underlying asset for a specific

price by a specific date and give the seller the obligation to buy the main asset for the same conditions if forced by the buyer. Each option contract corresponds to 100 units of the underlying asset, for example, 100 shares of stock. [5]

The price of an option can be found by using the Black-Scholes-Merton option pricing model [6], [7]. This model takes into account multiple factors to determine an option's price. The variable factors for this model are:

- Strike price
- Current underlying asset price
- Risk-free interest rate
- Days to expiration (DTE)
- Volatility of the price of the underlying asset
- Dividend yield of the underlying asset

An option trader can typically benefit the most by correctly predicting the underlying asset's price movement or volatility.

The Black-Scholes-Merton model defines the rates of change of the option price with respect to each variable. These are known as the option *greeks*. The main greeks are defined as:

- Delta - Represents the rate of change of the option price with respect to the underlying asset price.
- Theta - Represents the rate of change of the option price with respect to the passage of time.
- Gamma - Represents the rate of change of the option's delta with respect to the underlying asset price.
- Vega - Represents the rate of change of the option price with respect to the underlying asset's price volatility.

Note that vega isn't an actual greek letter. It is represented by the greek letter (ν), which resembles the letter "v". The option greek delta is generally used to indicate

the *moneyness* of an option, as defined below. An option can either be bought or sold (known as opening a trade). Therefore, there are four cases when opening a trade:

1. Buying a call, known as being "long on a call." A premium is debited to enter the trade.
   (a) Profit is achieved when the underlying asset price rises farther than the strike price, plus the premium paid. Maximum profit is theoretically unlimited.
   (b) Otherwise, a loss occurs. The maximum loss is the premium paid.
2. Selling a call, known as being "short on a call." A premium is credited to enter the trade.
   (a) Profit is achieved when the underlying asset price fails to rise farther than strike price, plus the premium received. The maximum profit is the premium received.
   (b) Otherwise, a loss occurs. Maximum loss is theoretically unlimited.
3. Buying a put, known as being "long on a put." A premium is debited to enter the trade.
   (a) Profit is achieved when the stock price falls below the strike price, minus the premium paid. The maximum profit occurs when the underlying asset price is \$0.
   (b) Otherwise, a loss occurs. The maximum loss is the premium paid.
4. Selling a put, known as being "short on a put." A premium is credited to enter the trade.
   (a) A profit is achieved when the stock price fails to fall below the strike price, minus the premium received. The maximum profit is the premium received.
   (b) Otherwise, a loss occurs. The maximum loss occurs when the underlying asset price is \$0.

Each of the possible option positions above has a different payoff, or Profit-Loss (P/L) graph (as shown in Figure 1.1). The P/L graphs for long and short stock positions are shown to contrast long and short strategies. The figures on the left column are long strategies, and the right column figures are short strategies.

Figure 1.1(a) shows the P/L of buying (going long) stock. In contrast, Figure 1.1(b) shows the P/L of selling (going short) stock. We assume a stock price of $300. Figures 1.1(c)-(f) correspond to the four option positions described above. We assume a strike price of $300.

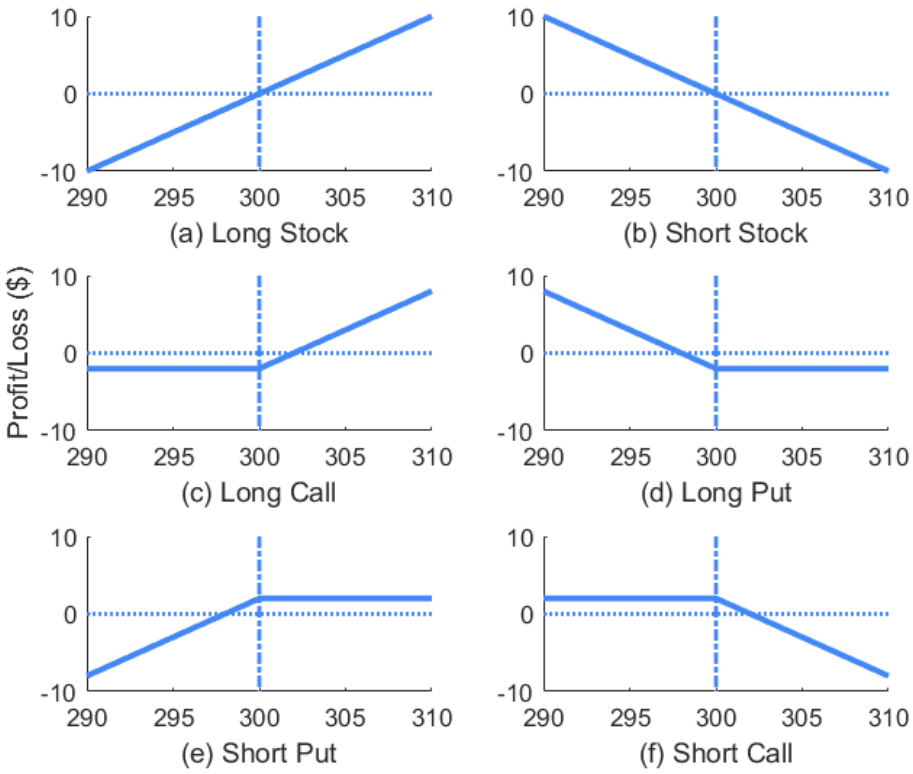

Figure 1.1: Example P/L payoff diagrams. Left column shows bullish (long) strategies, right column shows bearish (short) strategies.

As seen in Figure 1.1(a), for a long stock purchased for $300 per share, any future stock price higher than $300 results in a profit, and any price lower than $300 represents a loss. The reverse situation happens in Figure 1.1(b), representing a short stock sold at a $300 per share price. For this short stock position, a future price lower (higher) than $300 represents a profit (loss). The option P/L graphs in Figures 1.1(c)-(f) have a strike price of $300, representing the point of inflection of the change in

the graph's slope. Figure 1.1(c) represents a long call position. It results in a profit when the stock price is further than the strike price, plus the premium paid. It results in a limited loss (the premium paid) when the stock price is below it. Figure 1.1(e) represents a short put position. It results in a limited profit (the premium received) when the stock price is further than the strike price minus the premium received, and a loss when the stock price is below it. These two positions profit when the stock price rises, also known as a bullish position.

Figure 1.1(d) represents a long put position. It results in a profit when the stock price is below the strike price, plus the premium paid. It results in a limited loss (the premium paid) when the stock price is above it. Figure 1.1(f) represents a short call position. It results in a limited profit (the premium received) when the stock price is below the strike price minus the premium received, and a loss when the stock price is above it. These two positions profit when the stock price falls, also known as a bearish position.

When the underlying asset price is at the option's strike price, it is considered at-the-money (ATM). When the price is on the graph's non-zero slope, it is considered in-the-money (ITM). Otherwise, it is considered out-of-the-money (OTM). This nomenclature is commonly referred to as the moneyness of the option.

An alternative way to represent moneyness is by using the options greek delta. Delta represents the rate of change of the option price with respect to changes in the underlying asset price. Absolute delta values range from [0, 1] for both call and put options. An absolute delta value of 0.5 represents an ATM option, an absolute delta value > 0.5 represents an ITM option, and an absolute delta value < 0.5 represents an OTM option. Long calls and short puts have positive delta values, in contrast with long puts and short calls that have negative delta values.

Another way to represent moneyness is by using a normalized strike value (NV).

This value is obtained by dividing the strike price by the underlying asset price. For a call option, an NV value of 1 is considered ATM, an NV value > 1 is considered OTM, and an NV value < 1 is considered ITM. For a put option, an NV value of 1 is considered ATM, an NV value < 1 is considered OTM, and an NV value > 1 is considered ITM. [8].

1.3 Broken Wing Butterfly Strategy

In general, any number of options positions can be entered in the market at the same time. The combination of positions is called an option strategy. These strategies can consist of a combination of calls and puts. They can achieve profit not only by changes in the underlying asset price, but for example, changes in volatility, minimal changes in the underlying asset price, or even by the passage of time. The broken wing butterfly (BWB) is a multi-option strategy that consists of 3 options of the same type (call or put).

The put BWB is a bullish strategy that has increased in popularity due to its minimal upside risk, flexibility, potential profits, and high-profit rate. The put BWB we analyzed in this thesis consists of:

- One long put close to ATM, or slightly ITM or OTM
- Two short puts at the same strike, typically further OTM
- One long put, far OTM

The difference between each strike is referred to as the wing widths of the BWB. Unlike an ordinary butterfly strategy [5], the wing width of a BWB is not equidistant.

This combination enables the buildup of profit with the passage of time, even when the long strikes are not ITM, as shown in Figure 1.2. For this figure, the trade was entered when the options had 41 days to expiration (DTE), and the underlying asset price was at \$304.21. The long put strikes are at \$318 and \$275, and the short

strike is at $304. The T+0 line shows the P/L graph when the strategy was entered. The T+26 P/L graph shows the potential profit after 26 days, and the T+37 P/L graph shows the potential profit after 37 days. If the closing price of the underlying asset is below the lower long put strike of $275, this strategy experiences its maximum loss. If the underlying asset's closing price is precisely at the short strike of $304, this strategy experiences its maximum profit.

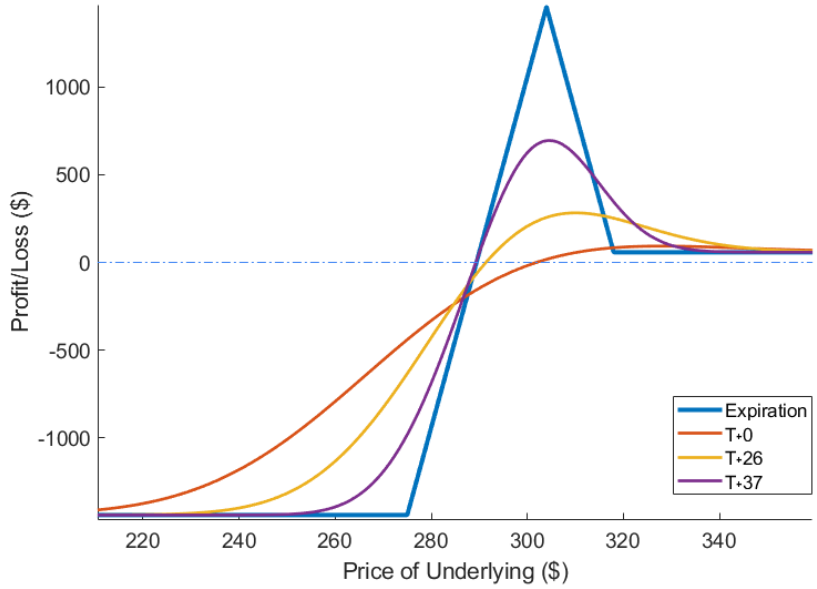

Figure 1.2: Broken wing butterfly structure. Expiration and theoretical P/L graphs for a strategy with 41 days to expiration. The T+0, T+26, and T+37 lines represent the current value of the trade, and the 26 and 37-day projections of the value of the trade, respectively.

There are several considerations to take into account when designing a BWB strategy. The following parameters can specify them:

- Days to Expiration (DTE)
- Maximum loss to exit a trade
- Minimum profit to exit a trade
- Exit days before expiration. This limits the number of days in a trade if other exit criteria have not been met

- Maximum debit or minimum credit to enter a trade. There is a maximum debit one is willing to pay or a minimum credit one is willing to receive to enter a trade
- Which strikes to choose for the four options

The values these parameters can take have wide ranges, making the search space to find an optimal set of parameters exponential. To solve this problem, in this thesis we used a differential evolution optimization method.

1.4 Differential Evolution

Differential evolution is an evolutionary algorithm that aims to optimize a problem using successive iterations to maximize desired properties, while minimizing undesired properties. It is commonly used for non-linear and non-differentiable continuous-space functions [9].

Differential evolution has been found to have better performance over other optimization techniques, including genetic algorithms, simple evolutionary algorithms, particle swarm optimization, for various functions that include real value parameters to optimize [4].

A differential evolution algorithm starts by randomly generating an initial population of solutions to a problem P_g. The population has a size NP, and each individual in the population is symbolized by the vector $\mathbf{x}$, of dimension D. Thus, a population at a given generation is defined as:

$$P_g = (\mathbf{x}_i), \quad i = 1, 2, 3, ..., NP, \quad g = 1, 2, 3, ..., g_{max} \tag{1.1}$$

$$\mathbf{x}_i = (x_{j,i}), \quad j = 1, 2, 3, ..., D \tag{1.2}$$

Where i is the ith vector member of the population P_g, j is the jth dimension of the vector $\mathbf{x}$, and g is the gth generation of the evolutionary algorithm, and $\mathbf{x} \in \mathbb{R}^D$.

This population evolves through mutation, crossover, and selection to find better "fit" individuals in the solution space. These operations are known as the DE operators, and they are discussed below.

1.4.1 Initialization

To initialize a population, upper and lower bounds are required. Generally, they are dependent on prior knowledge of the problem. Ideally, these boundaries would encompass only valid solutions to the problem in order to avoid the generation of invalid solutions and to constraint the search to a smaller search space. A uniform random number generator is typically used to generate each component of a population's vector. If we define the upper bound as B_u and the lower bound as B_l, then:

$$x_{j,i} = rand_j(0,1) \times (B_{j,u} - B_{j,l}) + B_{j,l} \tag{1.3}$$

Where $rand_j(0,1)$ represents a uniformly distributed number between $[0,1]$. Each component of the vector $\mathbf{x}$ can be bounded by different values.

1.4.2 Differential Mutation

In order to explore the fitness space for enhanced solutions, differential evolution uses the mutation step to mutate and recombine population members to create a population of mutant vectors of size NP using:

$$\mathbf{v}_{i,g} = \mathbf{x}_{r0,g} + F \times (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}) \tag{1.4}$$

Where F is a positive real number that controls the rate of mutation, usually it is set within the range (0, 2) [4]. $\mathbf{x}_{r0}$, $\mathbf{x}_{r1}$, and $\mathbf{x}_{r2}$ are randomly chosen vectors out of the current generation's population where $r_0, r_1, r_2 \in \{1, 2, ..., NP\}$ and are mutually

different.

Due to this operation, some of the child vector's components might be outside the defined bounds. For that reason, an extra step is necessary to ensure that each component is bounded. In this thesis, instead of forcing out of bound values to their closest boundary, we used the original vector to get a new component value by using:

$$v_{j,i,g} = \begin{cases} B_{j,u} + rand_j(0,1) \times (x_{j,i,g} - B_{j,u}), & \text{if } v_{j,i,g} > B_{j,u} \\ B_{j,l} + rand_j(0,1) \times (x_{j,i,g} - B_{j,l}), & \text{if } v_{j,i,g} < B_{j,l} \end{cases} \tag{1.5}$$

1.4.3 Crossover

Differential evolution generally implements a uniform crossover technique to create new population members, known as children. The children are symbolized by the trial vector $\mathbf{u}$ where:

$$\mathbf{u}_{i,g} = u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } rand_j(0,1) \leq Cr \text{ or } j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases} \tag{1.6}$$

Where $Cr \in [0,1]$ is the crossover probability. The second condition $j = j_{rand}$, $j_{rand} \in 1, 2, ..., D$ ensures that at least one component in the trial vector $\mathbf{u}$ is different than the target vector.

1.4.4 Selection

The next generation's members are selected using a greedy strategy. If the trial vector $\mathbf{u}_{i,g}$ has an equal or greater fitness value than the target vector, it takes the place of the target vector for the next generation. The next generation's population is given

by:

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g}, & \text{if } f(\mathbf{u}_{i,g}) \geq f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g}, & \text{otherwise} \end{cases} \tag{1.7}$$

Where f denotes the fitness function.

1.4.5 Convergence & Results

As generations increase, the population keeps improving and converging to the maximum point in the fitness space it can find. The algorithm stops either by a prespecified number of generations or after a certain number of generations where the maximum fitness does not improve.

This type of differential evolution is referred to as "DE/rand/1/bin" since it randomly selects the base vector for mutation. It only uses one vector difference for the mutation and uses a binomial distribution for selecting the new population.

Algorithm 1: A differential evolution algorithm

```
1  g → 1        ▷ initialize the generation counter
2  Generate a random initial population P_g of size NP
3  Compute the fitness, f(x_i), of every x_i ∈ P_g
4  while g ≤ g_max do
5      for each individual i in P_g do
6          Get random individuals x_r0, x_r1, x_r2 without replacement from P_g
7          Perform differential mutation on these vectors to create a mutant
             vector v_i
8          Perform crossover to create a trial vector u_i
9          Compute the fitness, f(u_i)
10         Select the member of the next generation's population P_{g+1} based on
             fitness
11     end
12     g → g + 1
13 end
```

1.5 Related Work

The use of evolutionary algorithms in the stock market has been primarily used to forecast stock prices or index levels with the help of neural networks [10], [11], [12] or to more accurately price options [13], [14].

Previous work done by Tymerski and Greenwood in [8] and [15] focused on using memetic algorithms to find unconventional optimal option strategies over a defined period. Their work found strategies that optimized for final profits while reducing drawdown. They used fixed entry parameters and always exited on the expiration of the options.

Optimizing the BWB was undertaken in [16]. They selected a limited range for each parameter and analyzed every permutation from 2014 to 2015 using the S&P 500 index options. However, they did not explore continuous ranges for these parameters.

Further work by Munoz Constantine, Tymerski, and Greenwood in [17] optimized the BWB options strategy with different mapping strategies for the whole period from 2005 to 2016. They found that the normalized strike mapping was superior to the other mapping methods and achieved better returns than merely holding the SPY stock.

This thesis most closely resembles the work of Munoz Constantine, Tymerski, and Greenwood's [17], but instead of optimizing the entire period, in this thesis we utilize in and out-of-sample periods to assess the suitability of differential evolution for predicting future optimal strategy parameters.

Chapter 2

Methods

2.1 Option Data

The underlying asset used for this thesis is the S&P 500 exchange-traded fund (ETF) that has the symbol SPY. The options data used for backtesting and optimization was the end of day (EOD) data obtained from IVolatility.com. This data spans from January 10, 2005, to July 15, 2016.

A sample of this data is shown in Table 2.1. It includes each option contract's bid price, ask price, the greek delta, expiration date, and the underlying's closing price. This data has been augmented to include the normalization value described in the subsequent sections of this thesis.

Date	Expiration Date	Call Bid	Call Ask	Call Delta	Put Bid	Put Ask	Put Delta	Close Price	Strike Price	Norm Value
8/5/2010	9/18/2010	$7.14	$7.19	0.7644	$1.47	$1.49	-0.2491	$112.85	$107	0.9482
8/5/2010	9/18/2010	$6.34	$6.39	0.7314	$1.66	$1.70	-0.2804	$112.85	$108	0.9570
8/5/2010	9/18/2010	$5.56	$5.62	0.6948	$1.91	$1.96	-0.3163	$112.85	$109	0.9658
8/5/2010	9/18/2010	$4.82	$4.88	0.6541	$2.22	$2.25	-0.3558	$112.85	$110	0.9747
8/5/2010	9/18/2010	$4.12	$4.17	0.6093	$2.52	$2.58	-0.3978	$112.85	$111	0.9836
8/5/2010	9/18/2010	$3.48	$3.52	0.5600	$2.91	$2.94	-0.4431	$112.85	$112	0.9924
8/5/2010	9/18/2010	$2.85	$2.91	0.5071	$3.32	$3.36	-0.4910	$112.85	$113	1.0013
8/5/2010	9/18/2010	$2.31	$2.37	0.4511	$3.80	$3.85	-0.5406	$112.85	$114	1.0102
8/5/2010	9/18/2010	$1.85	$1.89	0.3937	$4.34	$4.40	-0.5905	$112.85	$115	1.0190
8/5/2010	9/18/2010	$1.43	$1.47	0.3355	$4.95	$5.02	-0.6390	$112.85	$116	1.0279
8/5/2010	9/18/2010	$1.07	$1.11	0.2781	$5.62	$5.70	-0.6848	$112.85	$117	1.0367
8/5/2010	9/18/2010	$0.79	$0.82	0.2249	$6.34	$6.47	-0.7258	$112.85	$118	1.0456
8/5/2010	9/18/2010	$0.56	$0.60	0.1769	$7.13	$7.27	-0.7619	$112.85	$119	1.0545

Table 2.1: SPY's end of day option chain on 08/05/2010. The last column represents the corresponding strike normalized value (NV).

2.2 Assumptions

2.2.1 Transaction Costs

Due to recent innovations in the stock market, transaction costs for options trading has decreased dramatically. It ranges from \$0 to \$0.65 per contract for some brokers. In this thesis we ignored transaction costs for simplicity and the relatively low transaction costs per trade.

To avoid unrealistic transaction prices, in this thesis we used the following formula when entering and exiting a trade based on its bid/ask spread:

$$\textit{transaction price} = \begin{cases} \frac{1}{3} \cdot \textit{bid} + \frac{2}{3} \cdot \textit{ask} & \text{if buying} \\ \frac{2}{3} \cdot \textit{bid} + \frac{1}{3} \cdot \textit{ask} & \text{if selling} \end{cases} \tag{2.1}$$

To avoid trading contracts with low volume, if the bid price of a contract is less than \$0.08, it is not traded. In addition, we assume that the option contracts are not exercised for the duration of the trade.

2.2.2 Sizing

Every trade is entered with the minimum number of contracts possible (for the BWB, it is four contracts. two for the long puts, and two for the short puts).

2.3 Implementation

In this thesis we used the "DE/rand/1/bin" strategy to find each of the BWB optimal parameters. The parameters are optimized for a fixed period (the in-sample months) and then used for a single trade in the future (the out-of-sample months). Once the out-of-sample trade is completed, the next in-sample period will include the oldest

out-of-sample month, and drop the oldest in-sample month. This process sweeps all the data in a sliding window approach. This process guarantees only the most recent data is taken into account for future predictions. Figure 2.1 highlights this process.

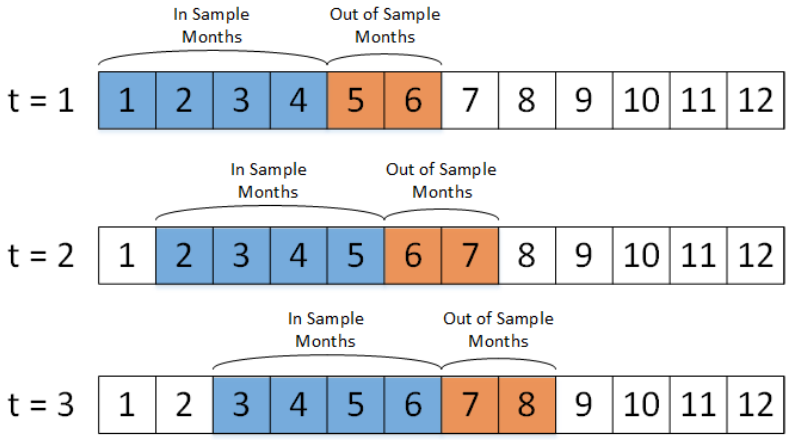

Figure 2.1: Sliding window of in and out-of-sample months at consecutive time steps.

The BWB parameters define how and when to enter and exit a trade. For in-sample months, trades are entered every day that the conditions are met. This condition ensures that as much data as possible is used to optimize the parameters.

For out-of-sample months, only one trade is entered for the whole period. If the entry conditions are not met, it skips to the next day until it finds a day satisfying all entry conditions. If no suitable trade is found, no trade is recorded for that period.

The trade parameters used, accompanied by sample values (using a points-based option's strike mapping to be discussed further below), are shown below.

1. Upper long strike, ULS, e.g. ULS = S+6, that is, 6 points above the underlying price, S.
2. Short strike, SS, e.g. SS = S-5, that is, 5 points below the underlying price, S.
3. Lower long strike, LLS, e.g. LLS = S-15, that is, 15 points below the underlying price, S.
4. Entry days to expiration, entry DTE = 45, e.g. 45 days to expiration at entry.

5. Exit days to expiration, exit DTE = 14, e.g. 14 days to expiration to exit.
6. Maximum cost to enter a trade, e.g. 10% of the required margin.
7. Minimum profit to exit a trade, e.g. 50% of the maximum possible profit.
8. Maximum loss to exit a trade, e.g. 40% of the required margin.

The sample values given above translate to a strategy that seeks to (by item 1) buy a long put 6 points above the underlying price, (by item 2) sell two short puts 5 points below the underlying price, and (from item 3) buy a long put 15 points below the underlying price. All put options are from an option chain with (by item 4) its days to expiration closest to 45 days. The total BWB strategy cost is limited to a maximum debit of (by item 6) 10% of the required margin.

This strategy will monitor the profit or loss of each day post-trade entry, and exit the trade when any one of the following three conditions occurs:

i. A profit target of 50% of the maximum possible profit is reached (by item 7).
ii. A maximum loss of 40% of the required margin is reached (by item 8).
iii. There are 14 days left until the expiration of the options (by item 5).

The fitness function to maximize was formulated to maximize risk-adjusted returns. It achieves this by optimizing the final profit and minimizing the maximum drawdown. It is a weighted sum of these terms given by:

$$fitness(\mathbf{x}) = \textit{final cumulative return}(\mathbf{x}) - \frac{\textit{maximum drawdown}(\mathbf{x})}{3} \tag{2.2}$$

Where $\mathbf{x}$ represents the vector of trade parameters. The first term adds the final cumulative returns of the strategy to ensure the total profits are maximized. The last term penalizes the fitness by an adjusted level of the maximum equity drawdown.

Besides maximizing the fitness value, the following conditions were required for a strategy to be valid.

a. Its fitness must be positive.

b. Its number of winning trades must be at least 70% of the total trades entered.

c. Its number of trades entered is at least 40% of the maximum possible trading days.

These requirements prevent (by item a.) the optimization of a losing strategy (by item b.) a few lucky trades to dominate the returns, and (by item c.) to have a strategy that trades infrequently. These conditions are required for the in-sample period only.

The initial parent population of the DE algorithm was generated by the following intelligent initialization method. Individuals in the population are randomly generated from a uniform distribution, their fitness is evaluated, and only individuals satisfying all three pre-conditions were added to the initial population. This process continues until all members of the population were generated.

After generating the initial population, the mutation and crossover operations were used to generate the children and explore the fitness landscape. These operations were repeated until the overall best individual did not change for several generations. The fittest individual out of all generations was kept. Subsequently, the whole process was repeated for 20 runs to explore the fitness landscape fully.

Once the best parameters were found for the in-sample period, those parameters were used for trading in the out-of-sample period. The next in and out-of-sample months were selected using the sliding window method described in Figure 2.1 until all the available data was explored.

Algorithm 2: My differential evolution implementation

```
r → 1      ▷ initialize the run counter
for r to 20 do
    g → 1      ▷ initialize the generation counter
    Generate an initial population P_g of size NP using intelligent
     initialization
    Compute the fitness, f(x_i), of every x_i ∈ P_g
    while best fitness has not changed for 75 generations do
        for each individual i in P_g do
            Get random individuals x_r0, x_r1, x_r2 without replacement from P_g
            Perform differential mutation on these vectors to create a mutant
             vector v_i
            Perform crossover to create a trial vector u_i
            Compute the fitness, f(u_i)
            Select the member of the next generation's population P_g+1 based
             on fitness
        end
        g → g + 1
    end
    Save best overall individual
end
Compute the out-of-sample fitness and trade metrics using the best in-sample
 individual
```

2.4 Implementation Details

2.4.1 Margin Requirement

Option margin refers to the capital required from an investor as collateral before they can initiate a trade. The CBOE provides a "margin manual"[18] that describes how to calculate the margin required for a regular trading margin account for well-known option strategies. This manual defines the formulas recommended for calculating the margin required for a traditional butterfly structure. However, it does not guide as to how to calculate the margin for a broken wing butterfly.

For non-conventional strategies that are risk-defined, most brokers require a margin equal to the maximum theoretical loss the strategy can experience. For undefined risk strategies, the margin calculation varies. See Appendix B for a complete explanation. Since the BWB is a risk defined strategy, the margin required can be calculated by examining its maximum theoretical loss.

The margin required for each BWB was calculated by inspecting their P/L graph at expiration. The maximum loss the BWB can experience at its expiration is its margin required. This is the lower level of the expiration line in the BWB P/L plot (left side of the plot in Figure 1.2).

2.4.2 Strike Mapping

Three parameters of the BWB specify the strikes of the long and short puts. From Table 2.1 we can observe that strikes are specified in a dollar amount. The strikes available are dependent on the closing price of the underlying. In Table 2.1, the closing price of the underlying is \$112.85 and it has strike prices from \$107 to \$119. If the underlying closing price was different, for example \$1000, then the strikes \$107 to \$119 might not be available for trading. Therefore, if we use a fixed dollar amount

to specify a strategy, it will not be stay consistent over time. Since the underlying price varies every day, a method that indirectly assigns strike prices was used. Three such methods are considered and evaluated for their efficacy. The three methods are:

1. Points Based Mapping.
2. Delta Based Mapping.
3. Normalized Strike Mapping.

Each of these mappings will be examined by referencing Table 2.1. This table shows a partial option chain for the SPY on the date 08/05/2010. The expiration date for these options is 09/18/2010, representing days to expiration (DTE) of 44 days. The underlying's (SPY) end of day closing price was \$112.85. Different strikes are shown with the corresponding bid and ask prices as well as their respective delta values. Note that the last column contains the values of each strike's normalized values (NV). For example, the strike price of \$113 is represented by an NV of approximately 1 (since \$113/\$112.85 $\approx$ 1). Let us consider, as an example, how the three mapping methods would represent the three strikes of \$118, \$113, and \$107.

Points based mapping: This mapping method assigns the strikes by using a relative point distance from the underlying's EOD closing price. If we denote the EOD closing price of \$112.85 as **S**, then each strike can be represented by as **S**+5.15, **S**+0.15, and **S**-5.85. (The nearest strike is chosen when evaluations do not result in exact strike prices). Option traders widely use this method.

Delta based mapping: This mapping method specifies strikes by using the absolute value of their corresponding delta values. Therefore, the above strikes can be specified as the 0.7, 0.5, and 0.25 delta strikes (Again, choosing the strike nearest to the delta value.)

Normalized Strike Mapping: This mapping method requires that the EOD closing

price normalize each strike in the option chain. Therefore, the strikes above can be referred via their normalized values of 1.045, 1.0, and 0.948 (the closest values are sufficient). This method was introduced in [8], and further explored in [17] where it was found to be highly effective in optimizing profits.

Chapter 3

Results

Results were generated by using a population size $NP = 25$, a mutation rate $F = 0.3$, and a crossover rate $Cr = 0.5$. The in-sample period spanned 12 months and the out-of-sample period one month. The fitness landscape exploration was stopped once the best in-sample fitness did not improve for 75 generations. This process was repeated for 20 runs. After the 20 runs, the next 12-month in-sample period is optimized in a sliding window fashion, as shown in Figure 2.1.

The value bounds used for each parameter are shown in Table 3.1 for all three strike mapping methods.

	Points Map	**Delta Map**	**Normalized Map**
Upper Long Strike, ULS	S-3 to S+40 (points)	strikes with 0.45 to 0.775 (delta)	strikes with NV 0.95 to NV 1.25
Short Strike, SS	S-35 to S+38 (points)	strikes with 0.05 to 0.75 (delta)	strikes with NV 0.73 to NV 1.24
Lower Long Strike, LLS	S-45 to S+29 (points)	strikes with 0 to 0.72 (delta)	strikes with NV 0.38 to NV 1.23
Entry DTE (days)	35 to 100	35 to 100	35 to 100
Exit DTE (days)	1 to 30	1 to 30	1 to 30
Max Cost (%)	-65 to 35	-65 to 35	-65 to 35
Min Profit (%)	0 to 100	0 to 100	0 to 100
Max Loss (%)	0 to 100	0 to 100	0 to 100

Table 3.1: Ranges are used for each parameter for each mapping type. The symbol S denotes the current underlying price. It is used in determining the strike locations for the points mapping method. The absolute delta values are shown, but in actuality, a long position will have a negative value, and a short position will have a positive delta value. NV represents normalized values.

In order to guarantee that the algorithm produces a BWB structure, the following requirement was added.

$$lower\ long\ strike, LLS < short\ strike, SS < upper\ long\ strike, ULS$$

The performance of the BWB strategy for all mapping methods is shown in Table 3.2. Out of the three mapping methods, the normalized strike mapping achieved the highest median out-of-sample fitness, with a value of 0.024. It achieved the best cumulative returns (of 151.71%), the highest Sharpe ratio (of 0.48), and the lowest drawdown (of 39.3%). The results of the other mapping methods are shown, as well as merely holding the SPY stock.

	Points Mapping	**Delta Mapping**	**Normalized Mapping**	**SPY**
Median In-Sample Fitness	5.22	4.75	4.88	1.04
Median Out-of-Sample Fitness	0.02	0.015	0.024	NA
Total Trades	95	101	97	1
Cumulative Returns	133.83%	98.53%	151.71%	122.70%
Annualized Returns	7.70%	6.10%	8.40%	7.20%
Annualized Volatility	21.70%	24.10%	22.0%	19.80%
Sharpe Ratio	0.45	0.37	0.48	0.45
Max Drawdown	41.0%	50.60%	39.30%	55.20%
Percent Profitable	61.05%	62.40%	63.0%	100%
Min Margin	\$737	\$657	\$417	\$54.77
Max Margin	\$4,268	\$4,054	\$4,813	\$198.40

Table 3.2: Performance comparison from 2005 to 2016. Margin results for the BWB are per tranche (1/2/1 contracts) and per share for SPY stock. The initial portfolio amount was \$10,000.

Figure 3.1 shows the correlation between the in-sample and out-of-sample fitnesses.

The linear regression shows a weak correlation for in-sample and out of sample fitnesses for all mapping methods. This suggests that the in-sample fitness does not predict whether an out-of-sample trade will be profitable.

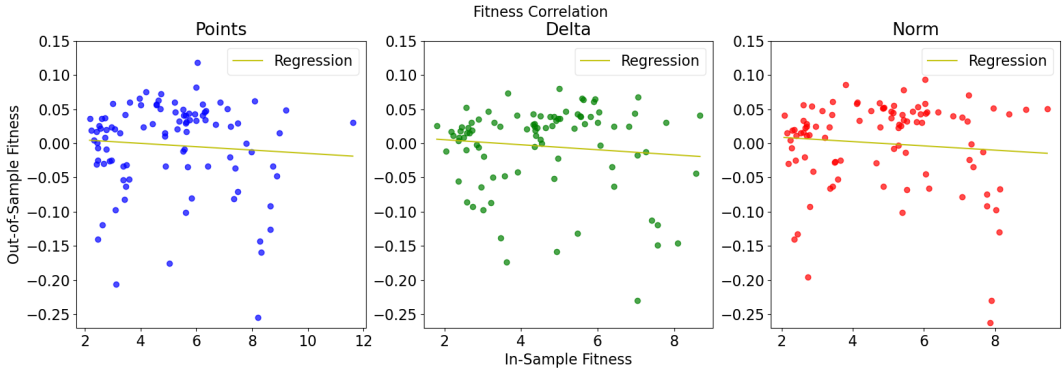

Figure 3.1: In-sample and out-of-sample fitness correlation. The linear regression of the variables shows a weak correlation.

Since the normalized mapping method achieved the best results, its optimal parameters are analyzed further in this section. See Appendix A for a complete comparison of the points and delta mapping methods.

Figure 3.2 shows the value ranges of the three strikes of the BWB over all in-sample periods. The upper long put strike always prefers to be ITM. The short puts spend most of the time ITM, except between 2008 to 2009, mid-2015 to 2016, and briefly in 2012. Interestingly, these periods coincide with periods of high volatility in the stock market, notably the great financial crash of 2008. The lower long put is consistently far OTM.

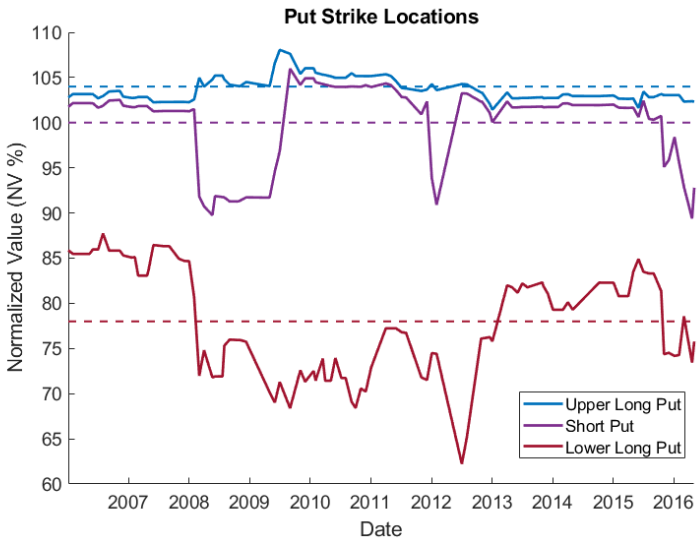

Figure 3.2: Optimal strike parameters for the normalized method. Upper long puts are, on average, ITM. Short puts are, on average, ATM. Lower long puts are, on average, far OTM. The dashed lines represent mean values.

From Figure 3.3, we can observe the optimum entry days to expiration is, on average, 90 days, and the exit days to expiration, on average, nine days. When the BWB strategy is near to expiration, the build-up of profit is more pronounced and more comfortable to achieve. It also increases the gamma risk of the strategy. Gamma is one of the option Greeks previously discussed. It refers to the higher sensitivity of option prices due to underlying price variations. The short exit days to expiration parameter helps to explain its high cumulative returns and elevated volatility.

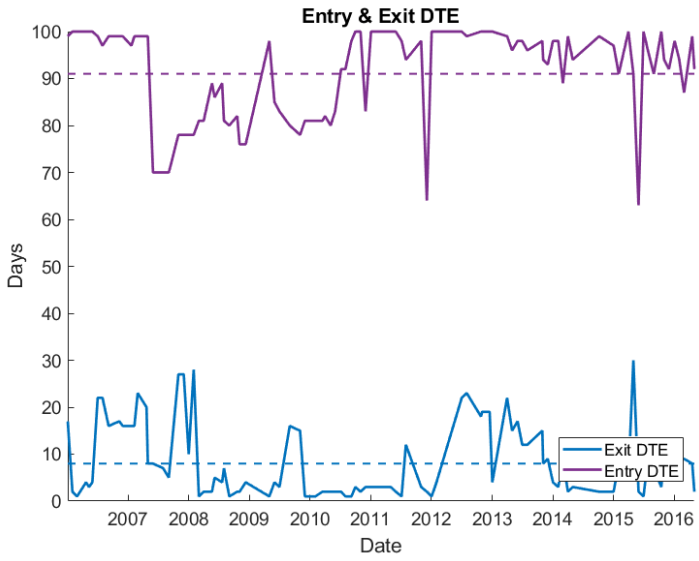

Figure 3.3: Optimal DTE parameters for the normalized method. Entry DTE on average, is 90 days, and exit DTE is nine days. The dashed lines represent mean values.

From Figure 3.4, we observe significant variability in the profit and loss taking parameters. On average, the strategy takes a more significant percentage of profit than of loss. This result, combined with the low exit days to expiration parameter, suggests that the optimization method is trying to predict where the price is going close at expiration to take the maximum profit possible. It also suggests it is willing to take more significant losses in the search for maximum profit. The maximum loss parameter is significantly reduced during the significant financial crisis period. This suggests that the optimization method learned to avoid significant losses during volatile periods.

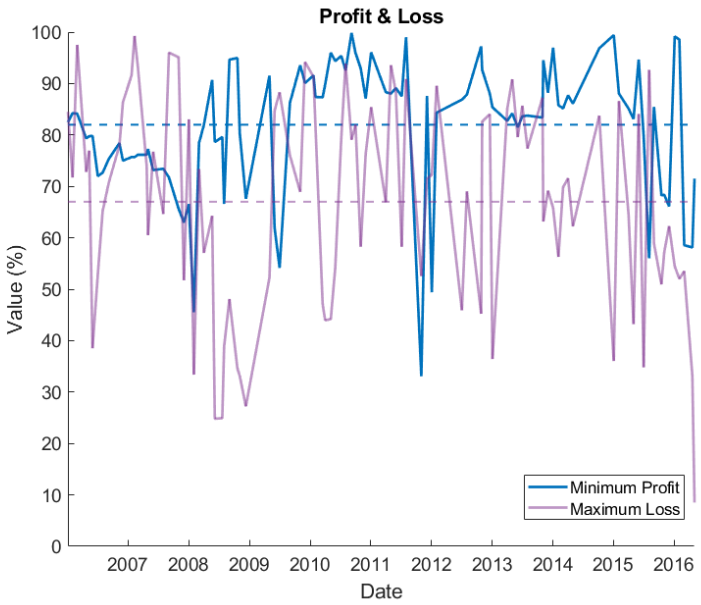

Figure 3.4: Optimal profit and loss parameters for the normalized method. It has higher average profit-taking than loss taking point. Dashed lines represent mean values.

Figure 3.5 shows great variation on the cost to enter a trade. On average, we are willing to pay to enter a trade.

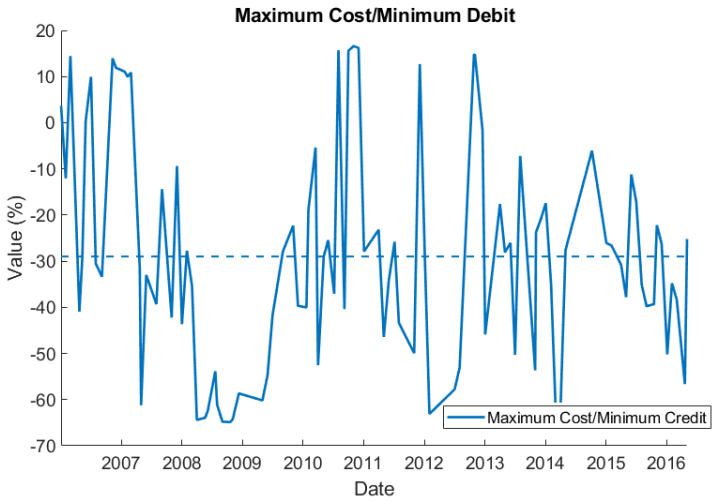

Figure 3.5: Optimal cost parameter for the normalized method. Mean value represented by dashed line.

Table 3.3 shows the percentage of times a specific exit condition was triggered

out of all trades. For all mapping methods, we observe that the exit parameter that influences the strategy's exits the most is the Exit DTE parameter. This demonstrates that the BWB strategy spends most of the time avoiding a maximum loss and obtaining a moderate profit or loss.

Both the points and delta mapping methods have higher maximum losses than the normalized method. Even though the points mapping method has a more significant percentage of maximum profit-taking, it is not enough to overcome its losses. This explains why the normalized and points mapping methods have higher cumulative returns than the delta mapping method.

As seen from Figure 3.3, the average days between entry and exit is close to 83 days. Therefore, as we get near the end of the period analyzed, there is not enough days to complete the trade. This is why we observe that 2% of the trades resulted in a forced exit.

This is the result of entering trades by the end of the period analyzed. The lack of data results in an inconclusive exit.

Exit Case	**Points Map**	**Delta Map**	**Normalized Map**
Max Loss (%)	10	9	6
Min Profit (%)	8	5	5
Exit DTE (%)	80	84	87
Forced Exit (%)	2	2	2

Table 3.3: Percentage of trades where each trade exit condition was triggered.

Figure 3.6 shows the equity curves for all mapping methods, along with the returns of the SPY stock. It shows that the normalized mapping method achieves the highest cumulative return and lowest drawdown. The points method achieves greater returns than the SPY, and the delta method has similar returns to the SPY. Figure 3.7 shows the beta factor each mapping method strategy has with its underlying. The beta factor is a measure of how much the underlying's returns explain the strategy's

returns. In general, a beta factor close to 0 is ideal since it means the strategy's returns are independent of the underlying movements. All mapping methods have a low mean beta exposure to the SPY.

Figure 3.8 show the returns of each strategy. It shows the greater volatility of the normalized mapping method and the delta mapping method's reduced volatility.

Figure 3.9 shows the annual returns for each method. All methods have a negative return on the same years, implying that during those years, there are events that are difficult to generalize.

Figure 3.10 shows the underwater plot of each mapping method. The underwater plot lets us visualize the drawdown's magnitude and duration. It shows that the normalized method's lower and shorter drawdown periods.

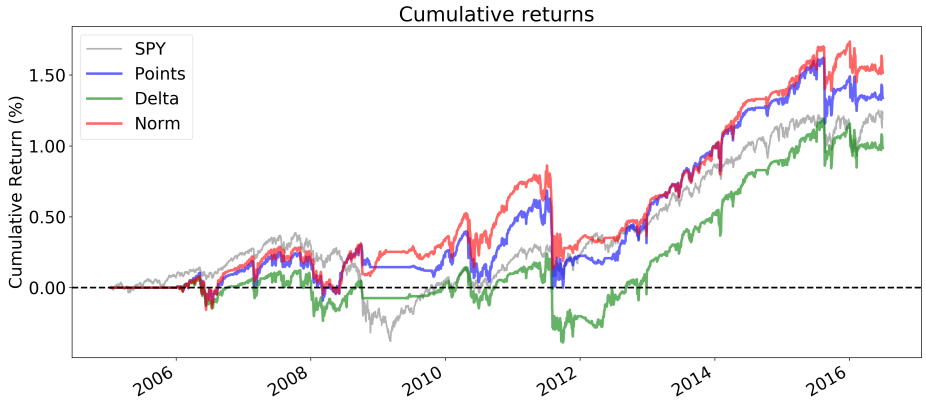

Figure 3.6: Cumulative returns comparison for all mapping methods and SPY.

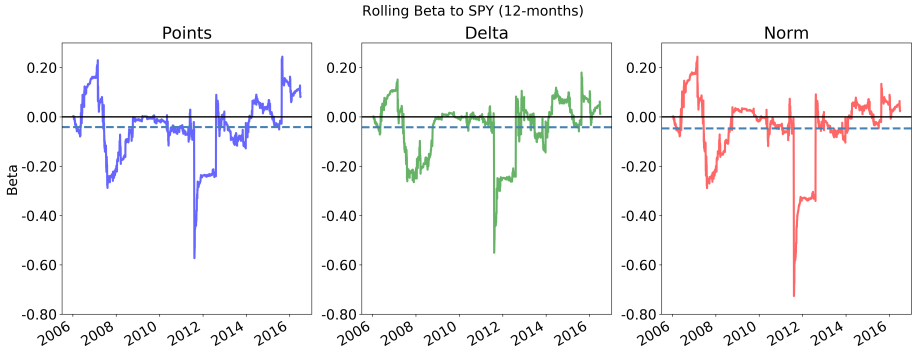

Figure 3.7: Beta factor to SPY comparison for all mapping methods.

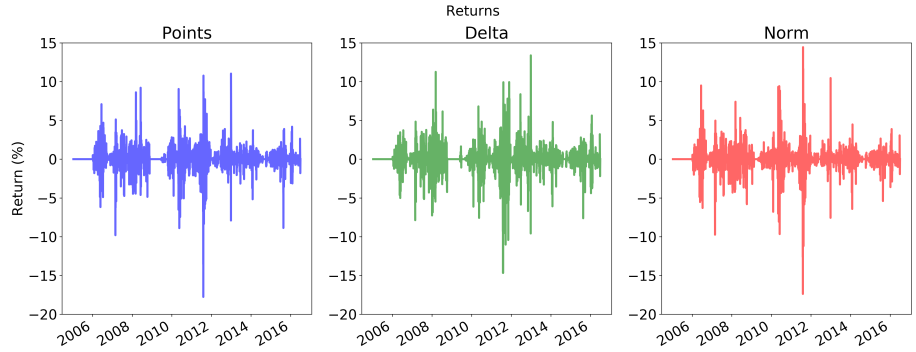

Figure 3.8: Returns comparison for all mapping methods.

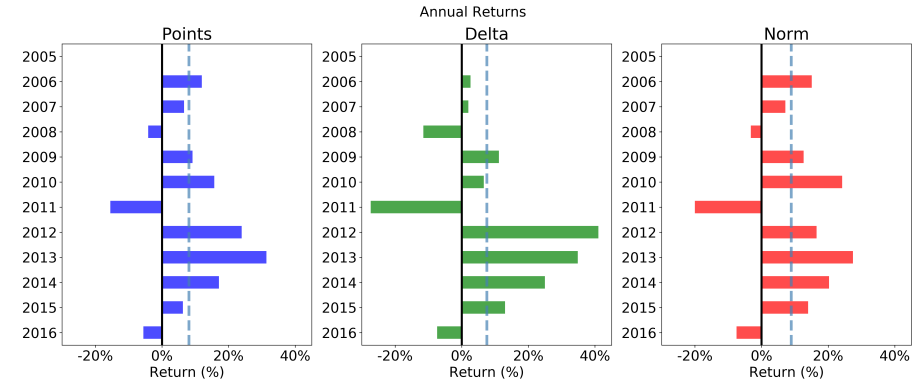

Figure 3.9: Annual returns comparison for all mapping methods.

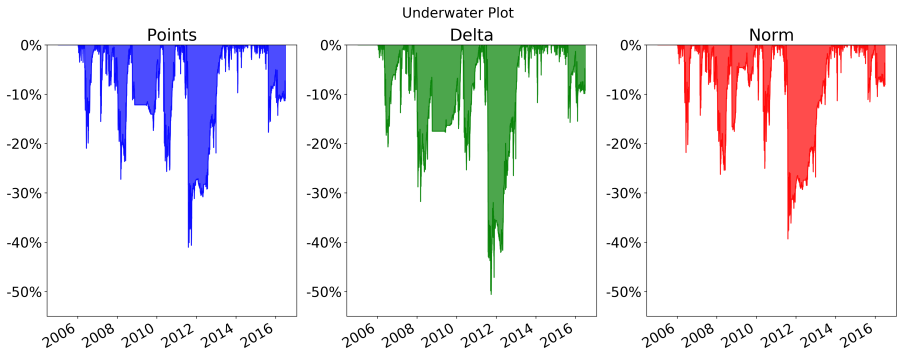

Figure 3.10: Underwater comparison for all mapping methods.

To verify that differential evolution is the best method to optimize the BWB parameters, in this thesis we evaluated different optimization methods to find the optimal parameters. These methods are included in Matlab's Global Optimization Toolbox. They include global search, particle swarm optimization, and simulated annealing. Out of all methods, only simulated annealing was able to find at least one valid solution for all mapping methods. Therefore, Table 3.4 contrasts the results against simulated annealing only. This table shows that the differential evolution method achieving better results for all mapping methods.

	Points Map DE	Points Map Sim. Annealing	Delta Map DE	Delta Map Sim. Annealing	Normalized Map DE	Normalized Map Sim. Annealing
Median In-Sample Fitness	5.22	1.95	4.75	0	4.88	0
Median Out-of-Sample Fitness	0.02	0.01	0.015	0	0.024	0
Total Trades	95	58	101	43	97	5
Cumulative Returns	133.83%	68.60%	98.53%	-6.7%	151.71%	4.70%
Annualized Returns	7.70%	4.60%	6.10%	-0.6%	8.40%	0.4%
Annualized Volatility	21.70%	21.0%	24.10%	5.0%	22.0%	1.7%
Sharpe Ratio	0.45	0.32	0.37	-0.09	0.48	0.24
Max Drawdown	41.0%	50.20%	50.60%	29.80%	39.30%	4.0%
Percent Profitable	61.05%	63.0 %	62.40%	65.0%	63.0%	40.0%
Min Margin	$737	$683	$657	$76.70	$417	$76.70
Max Margin	$4,268	$4,636	$4,054	$4,720	$4,813	$4,720

Table 3.4: Performance comparison from 2006 to 2016 using DE and simulated annealing optimization methods. Margin results for the BWB are per tranche (1/2/1 contracts) and per share of SPY stock. The initial amount was $10,000.

Chapter 4

Conclusions & Future Work

4.1 Conclusions

In this thesis, we examined the optimal parameters for the BWB option strategy. In particular, the predictive power of past optimal parameters for future trades was assessed. The optimal entry parameters consist of the three strike locations, days to expiration to entry, and adequate debit or credit limits. Optimal exit parameters involve minimum profit taking level, maximum loss level, and days to expiration to exit.

The use of a differential evolution algorithm was useful in finding the optimal parameters over other optimization methods. The use of the found optimal parameters for future trades generated profit metrics superior to its underlying. The issue of strike mapping, in which an indirect method is used to select option strikes, was addressed. The normalized strike mapping method was found to be the most effective of the three methods considered. As shown in the cumulative returns plot in Figure 3.6, all methods exhibit the same characteristics at each period. However, the normalized mapping method achieves better results, confirming its superiority as an indirect mapping method. This method represents strikes as normalized values with respect to their current underlying price. The normalized mapping method achieves the best cumulative return, lowest drawdown, and highest Sharpe ratio out of all mapping methods.

Figures 3.3-3.5, A.2-A.4, and A.6-A.8 show that the mean parameters for all mapping methods are similar.

Figures 3.2, A.1, and A.5 show great variability on the strike locations over time. Noting how the methods do not agree when to have the short strike ITM or OTM, but on average, they have the upper long put strike ITM, the short put strike closer to ATM, and the lower long put strike far OTM. This helps explain why the methods differ in their performance even though the other optimal parameters are similar.

Figures 3.3, A.2, and A.6 show that all methods have an optimal average entry DTE between 88 and 91, and an average exit DTE between 8 to 9.

Figures 3.4, A.3, and A.7 show that all methods have an optimal profit taking level between 82% and 90%, and an optimal loss taking level of 67% and 71%. This implies that taking profits and losses early helps limiting the drawdown and volatility while still achieving high returns (see Tables 3.2 and 3.3).

Figures 3.5, A.4, and A.8 show that on average we can tolerate a debit between 26% and 29% of the total required margin to enter a trade.

The optimal parameters' similarity further confirms that all the mapping methods are learning similar features from the underlying's movements. The selection of the option strikes makes a significant difference in the final profits and performance of a strategy.

All mapping methods achieve an average low beta exposure to the SPY, making it ideal for achieving returns in different market conditions.

The final performance of the normalized mapping method outperformed the other methods and of its underlying. It has a greater cumulative return, lower drawdown, comparable volatility, and higher Sharpe ratio while keeping its beta exposure to a minimum. Using differential evolution to optimize entry and exit parameters for the BWB option strategy proved useful in forecasting favorable entry and exit parameters

for trades.

4.2 Future Work

The normalized strike selection effectively transformed the BWB structure to a loss limited put ratio spread. Typically, the BWB has a 60%/40% wingspan ratio. However, over the decade of option data that the strategy was optimized, on average, the ratio spread type structure was optimal, except on periods of high volatility. Future work will focus on limiting the useful delta of the structure. Since delta changes through time, an adjustment frequency trade should be explored to keep the required delta within the limited range. This will have the effect of stabilizing the BWB wingspan and reducing its overall risk. However, this can have the effect of reducing overall profit. Therefore a new fitness function will have to be crafted to take into account this risk.

Bibliography

[1] J. A. Sarkett, "Road Trip With Options Supertraders," Stocks & Commodities V. 35:02 (22–27).

[2] J. Locke, "M3 Trade Simplified AND Improved In The M3.4u," 2019, https://www.lockeinyoursuccess.com/m3-trade-simplified-and-improved-in-the-m3-4u.

[3] SMB Capital, "Netzero Options," 2020, https://www.smbtraining.com/blog/netzero-options.

[4] K. Price, R. Storn, J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization.* New York: Springer-Verlag, 2005.

[5] J. Hull, *Options, Futures, and Other Derivatives.* New York: Pearson, 2015.

[6] F. Black, M. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy,* vol. 81, no. 3, pp. 637-654, May-Jun. 1973.

[7] R. Merton, "Theory of Rational Option Pricing," *The Bell Journal of Economics and Management Science,* vol. 4, no. 1, pp. 141-183, 1973.

[8] R. Tymerski, G. Greenwood, "Designing Equity Option Strategies Using Memetic Algorithms," *Technology and Investment,* vol. 9, no. 4, pp. 179-202, Nov. 2018.

[9] R. Storn, K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization,* vol. 11, pp. 341-359, Dec. 1997.

[10] M. Abdual-Salam, H. Abdul-Kader, W. Abdel-Wahed, "Comparative Study between Differential Evolution and Particle Swarm Optimization Algorithms in Training of Feed-Forward Neural Network for Stock Price prediction," in *The 7th International Conference on Informatics and Systems,* 2010.

[11] A. K. Route, P. K. Dash, R. Dash, R. Bisoi, "Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach," *ournal of King Saud University - Computer and Information Sciences,* vol. 29, no. 4, pp. 536-552, Oct. 2017.

[12] E. Nourani, A. Rahmani, A. Navin, "Forecasting stock prices using a hybrid Artificial Bee Colony based neural network," in *International Conference on Innovation, Management and Technology Research,* pp. 486-490, 2012.

[13] X. Gong, X. Zhuang, "Option pricing for stochastic volatility model with infinite activity Lévy jumps," *Physica A: Statistical Mechanics and its Applications,* vol. 455, pp. 1-10, Nov. 2015.

[14] J. Yao, Y. Li, C. Tan, "Option price forecasting using neural networks," *Omega,* vol. 25, no. 4, pp. 455-466, Aug. 2000.

[15] R. Tymerski, G. Greenwood, D. Sills, "Equity Option Strategy Discovery and Optimization Using a Memetic Algorithm," *Artificial Life and Computational Intelligence,* vol. 10142, Dec. 2016.

[16] D. L. Wilt, "Options Trade Evaluation Case Study," IOTA Technologies, Jan 2016. Accessed from the internet Jan 2016.

[17] D. Munoz Constantine, R. Tymerski, G. Greenwood, "Differential Evolution Optimization of the Broken Wing Butterfly Option Strategy," *Technology and Investment,* vol. 11, no. 3, pp. 23-45, Jul. 2020.

[18] Chicago Board Options Exchange, *Margin Manual*, CBOE, April, 2000. http://www.cboe.com/LearnCenter/pdf/margin2-00.pdf.

[19] Tastyworks Inc., *Technology*. https://tastyworks.com/technology.

[20] TD Ameritrade Inc., *Trading Platforms*. https://www.tdameritrade.com/tools-and-platforms/thinkorswim/desktop/download.page.

Appendix A

Optimal Parameters of Remaining Mapping Methods

A.1 Points Mapping

Figure A.1 shows the value ranges of the three strikes of the BWB over all in-sample periods. As with the normalized method, the upper long put strike always prefers to be ITM. The short puts spend most of the time ITM, except between 2008 to 2009, mid-2015 to 2016, and briefly in 2012. This confirms that these two methods are learning similar strategies.

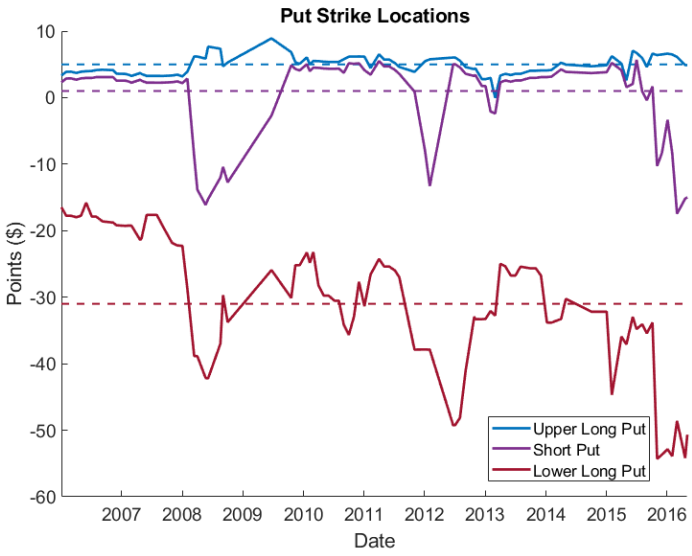

Figure A.1: Optimal strike parameters for the points method. Upper long puts are, on average, ITM. Short puts are, on average, ATM. Lower long puts are, on average, far OTM. The dashed lines represent mean values.

From Figure A.2, we can observe the optimum entry days to expiration is, on

average, 91 days, and the exit days to expiration, on average, eight days. Coinciding with the normalized method optimum values. The gamma risk on this strategy is increased due to the low exit DTE.

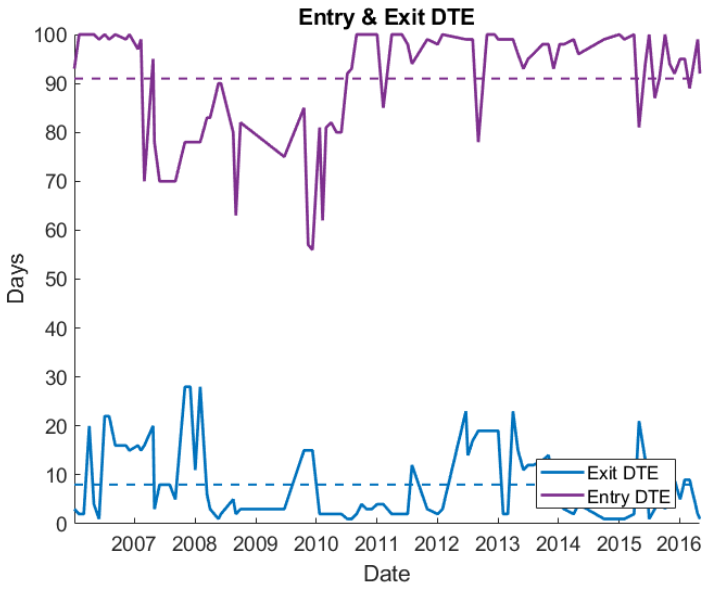

Figure A.2: Optimal DTE parameters for the points method. Entry DTE on average, is 91 days, and exit DTE is eight days. The dashed lines represent mean values.

From Figure A.3, we observe significant variability in the profit and loss taking parameters. On average, the strategy takes a more significant percentage of profit than of loss.

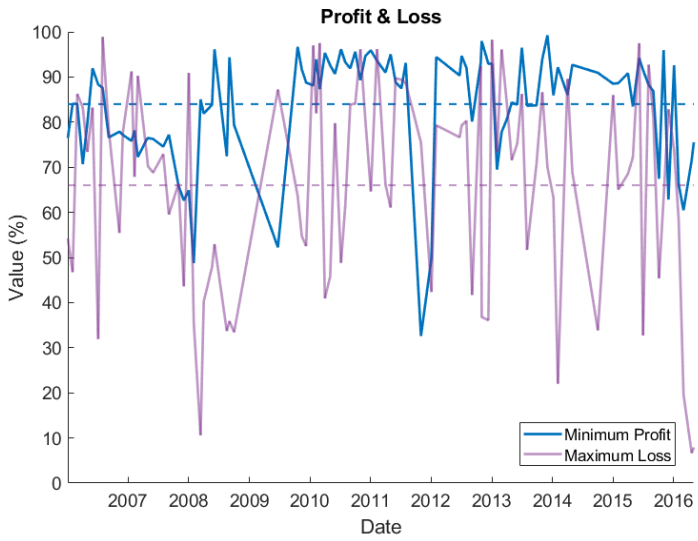

Figure A.3: Optimal profit and loss parameters for the points method. It has higher average profit-taking than loss taking point. Dashed lines represent mean values.

Figure A.4 shows great variation on the cost to enter a trade. On average, we are willing to pay to enter a trade. All of these parameters are similar to the found in the normalized method.

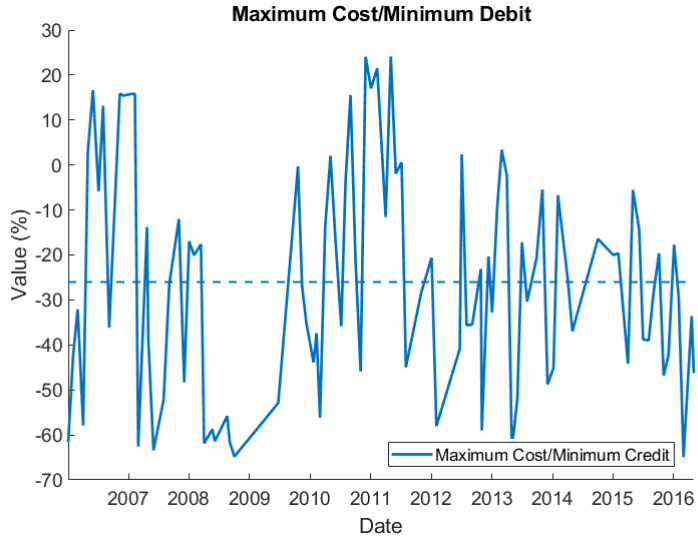

Figure A.4: Optimal cost parameter for the points method. Mean value represented by dashed line.

A.2 Delta Mapping

Figure A.5 shows the value ranges of the three strikes of the BWB over all in-sample periods. As with the normalized and points methods, the upper long put strike always prefers to be ITM. The short puts spend most of the time ITM, except between 2008 to 2009, mid-2015 to 2016, and briefly in 2012. This confirms that all methods are learning similar strategies.

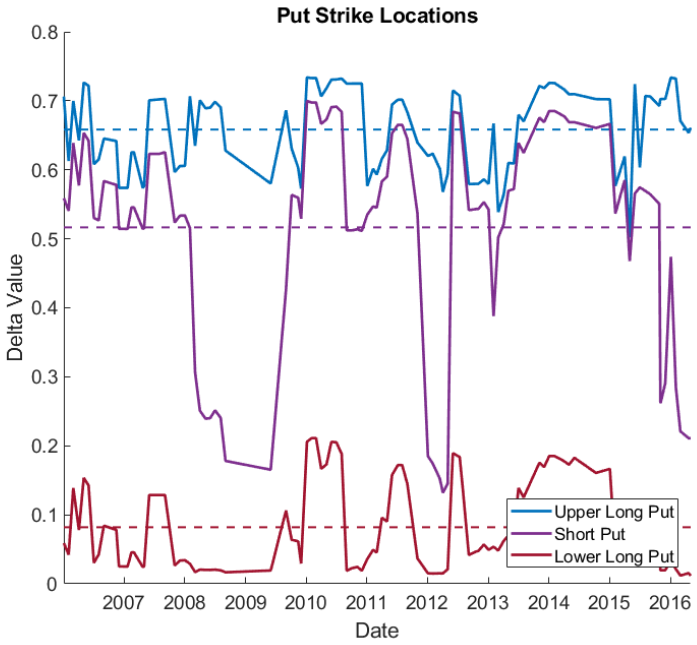

Figure A.5: Optimal strike parameters for the delta method. Upper long puts are, on average, ITM. Short puts are, on average, ATM. Lower long puts are, on average, far OTM. The dashed lines represent mean values.

From Figure A.6, we can observe the optimum entry days to expiration is, on average, 88 days, and the exit days to expiration, on average, eight days. Coinciding with the normalized and points methods optimum values. This suggests an optimal entry point overall.

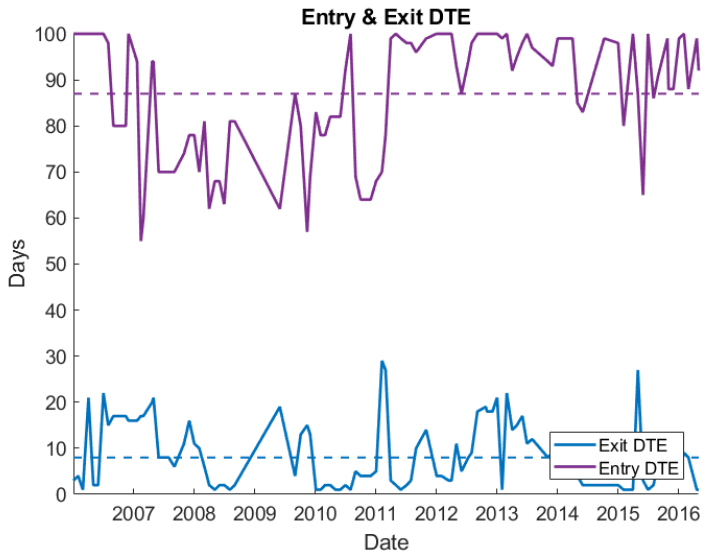

Figure A.6: Optimal DTE parameters for the delta method. Entry DTE on average, is 91 days, and exit DTE is eight days. The dashed lines represent mean values.

From Figure A.7, we observe as well significant variability in the profit and loss taking parameters. On average, the strategy takes a more significant percentage of profit than of loss.

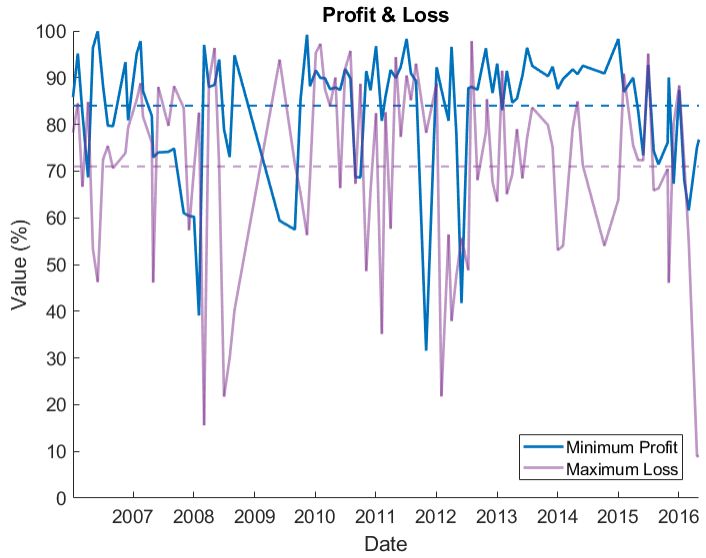

Figure A.7: Optimal profit and loss parameters for the delta method. It has higher average profit-taking than loss taking point. Dashed lines represent mean values..

Figure A.8 shows great variation on the cost to enter a trade. On average, we are willing to pay to enter a trade. All of these parameters are similar to the found in the normalized and points methods.

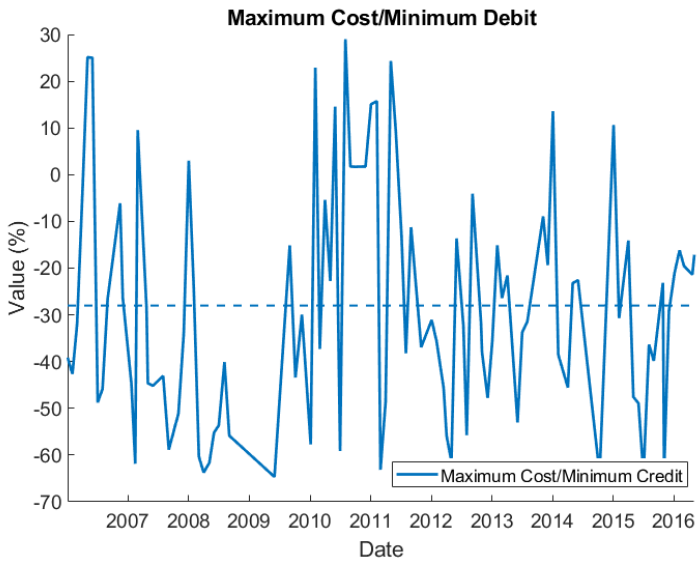

Figure A.8: Optimal cost parameter for the delta method. Mean value represented by dashed line.

Appendix B

Margin Calculation for Any Options Strategy

As previously discussed, for any defined loss options strategy, the margin required by most brokers in a regular account is the maximum loss the strategy can incur. However, when the strategy is undefined, there is no clear guidance on calculating the required margin, and brokers sometimes disagree on the final margin cost. As an example, let us take the SPY options chain on 6/18/2020, as seen in Table B.1

Date	Expiration Date	Call Bid	Call Ask	Call Delta	Put Bid	Put Ask	Put Delta	Close Price	Strike Price
6/18/2020	7/17/2020	$13.52	$13.61	0.66	$7.63	$7.68	-0.36	$311.78	$304
6/18/2020	7/17/2020	$8.43	$8.48	0.53	$10.49	$10.58	-0.47	$311.78	$312
6/18/2020	7/17/2020	$5.33	$5.39	0.42	$13.37	$13.50	-0.57	$311.78	$318
6/18/2020	7/17/2020	$4.48	$4.53	0.38	$14.46	$14.65	-0.60	$311.78	$320

Table B.1: SPY EOD option chain data for the date 6/18/2020.

If we sell one $304 strike put, two $318 strike calls, one $320 strike call, and buy one $312 strike call, we have an undefined strategy that resembles the most to a traditional strangle strategy. The P/L graph of this strategy is shown in Figure B.1.

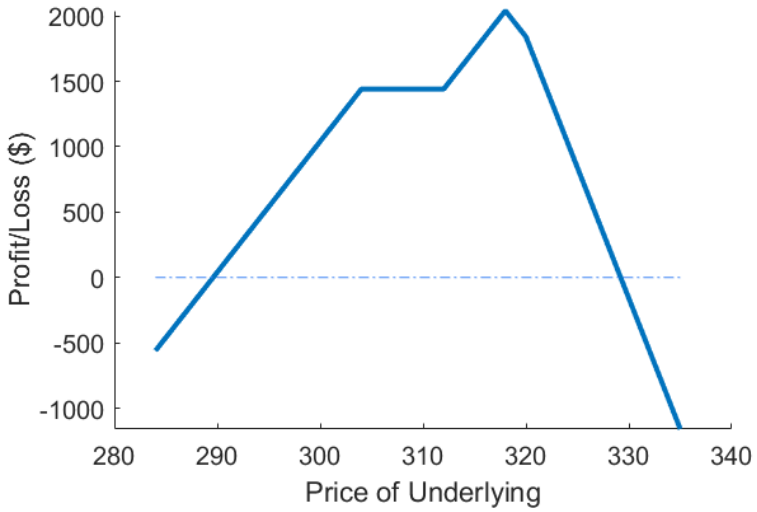

Figure B.1: Expiration P/L of the example's non-conventional strategy. Last underlying price was $311.78.

Two popular brokers, Tastyworks [19] and TD Ameritrade [20], show different margin requirements for the same strategy in their respective trading platforms. Figure B.2 shows that Tastyworks requires a margin of $11,163.98 (after fees), and Figure B.3 shows that TD Ameritrade requires a margin of $11,401.13 (after fees). These discrepancies are not vast, but enough to suggest that each broker uses different methods to calculate the margin required for the same strategy.

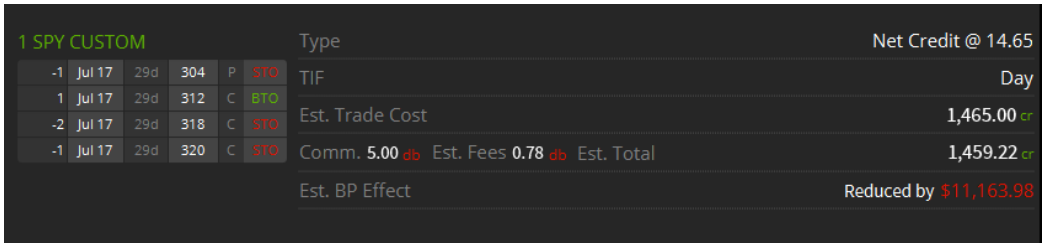

Figure B.2: Margin required by Tastyworks for our non-conventional strategy. Represented by the "Est. BP Effect" field.

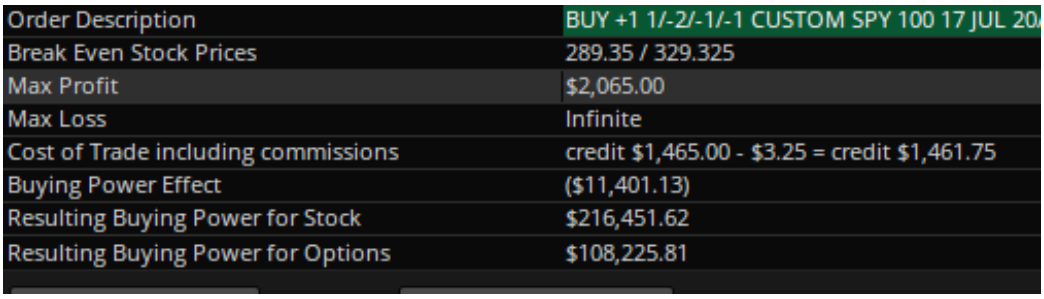

Figure B.3: Margin required by TD Ameritrade for our non-conventional strategy. Represented by the "Buying Power Effect" field.

In this thesis we propose a simple approach to calculate the margin for any unconventional and undefined loss strategy. We start by decomposing the strategy into its most basic components. The CBOE defines the following strategies for options with the same expiration date:

- Iron Condor (4 contracts).
- Butterfly (3 contracts).
- Spreads (2 contracts).
- Straddle/Strangle (2 contracts).
- Single options (1 contract).

Any strategy can be decomposed into these more straightforward strategies. The benefits of using these strategies are that they are more margin efficient, and the CBOE provides exact formulas to calculate their margins.

To calculate the margin of any strategy, we start by trying to find these embedded strategies, starting with the strategies with the most number of contracts first. For our example strategy above, we first try to check if we have the correct contracts to create an iron condor strategy. Since we do not, we check if we have the correct contracts to create a butterfly, and so on. In the end, we will have a collection of them. For our example strategy, we find that:

- One Long spread. Formed by using the $312 strike long call and $318 strike short call. It requires $311 in margin.
- One Strangle. Formed by using the $304 strike short put and $318 strike short call. It requires $5,768.60 in margin.
- One Short option. Formed by using the $320 strike short call. It requires $5,413.60 in margin.

This approach results in a total margin of $11,182.20 (found by adding the margin required for all embedded strategies). This margin value most closely matches Tastyworks' required margin.

This approach works best when the option strategy has an undefined loss. If a strategy has a defined loss (as is the BWB), its margin is found by examining the maximum loss at the strategy's expiration.

Appendix C

Source Code

```
clear
clc
close all
format bank
%initialize variables
global tradingDaysCell;
global expirationDaysCell;
global expirationDays;
global ivRank;
global tradingDays;
global vix;
%load data
load('opts_breakdown_vix.mat');
vix = vix(find(vix(:,1) == tradingDays(1)):end,5);

global NORM_MAPPING;
global DELTA_MAPPING;
global POINTS_MAPPING;

NORM_MAPPING = 1;
DELTA_MAPPING = 2;
POINTS_MAPPING = 3;
%select the type of mapping for the BWB strikes
mappingType = POINTS_MAPPING;

%type of mapping, can be delta, norm, or points
if mappingType == POINTS_MAPPING
  %ulsRange is how many points above or below ATM
  %Upper long strike, how many points above or below current SPY price
  ulsRange = [-3 42];
  %ssRange is how many points below uls
  %Short strike, how many points below uls
  ssRange = [1 32];
  %llsRange is how many points below Short Strike
  %Lower long strike, how many points below short strike
  llsRange = [10 60];
  %mpaRange is how many points above upper long strike to trigger below condition
  %Time exit: DTE unless mpaRange points above upper long strike
  mpaRange = [1 35];
  %minimum deviation to tolerate when it cannot find an exact option in
find_chains_params.m
  Options.minThresh = 2;
  OptionsOS.minThresh = 2;
elseif mappingType == NORM_MAPPING
  %ulsRange is how many percentage points above ATM. If uls = 0.02 then it is
equivalent to 1.02 (ITM)
  %Upper long strike, how many percentage points above current SPY price
  ulsRange = [-0.05 0.25];
  %ssRange is how many percentage points below ATM. If ss = 0.02 then it is equivalent
to 0.98 (OTM)
  %Short strike, how many percentage points below current SPY price
  ssRange = [0.01 0.22];
  %llsRange is how many percentage points below short strike. If lls = 0.03 then it is
equivalent to (0.98-0.03)=0.95 (OTM)
  %Lower long strike, how many percentage points below short strike
  llsRange = [0.01 0.55];
  %mpaRange is how many percentage points above upper long strike to trigger below
condition. if mpa = 0.05 then it is equivalen to (1.02+0.05)=1.07 (ITM)
  %Time exit: DTE unless mpaRange percentage points above upper long strike
  mpaRange = [0.01 0.35];
  %minimum deviation to tolerate when it cannot find an exact option in
find_chains_params.m
  Options.minThresh = 0.005;
  OptionsOS.minThresh = 0.005;
```

```
59   elseif mappingType == DELTA_MAPPING
60     %ulsRange is how many delta points above ATM. If uls = 0.02 then it is equivalent to
     -0.52 (ITM)
61     %Upper long strike, how many delta points above current SPY price
62     ulsRange = [-0.05 0.275];
63     %ssRange is how many delta points below ATM. If ss = 0.02 then it is equivalent to
     -0.48 (OTM)
64     %Short strike, how many delta points below current SPY price
65     ssRange = [0.023 0.45];
66     %llsRange is how many delta points below short strike. If lls = 0.03 then it is
     equivalent to (-0.48+0.03)=-0.45 (OTM)
67     %Lower long strike, how many delta points below short strike
68     llsRange = [0.03 0.5];
69     %mpaRange is how many delta points above upper long strike to trigger below
     condition. if mpa = 0.05 then it is equivalen to (-0.52-0.05)=-0.57 (ITM)
70     %Time exit: DTE unless mpaRange delta points above upper long strike
71     mpaRange = [0 1];
72     %minimum deviation to tolerate when it cannot find an exact option in
     find_chains_params.m
73     Options.minThresh = 0.035;
74     OptionsOS.minThresh = 0.035;
75   end
76   %percentage points relative to margin required. If margin in a specific trade is
77   %1000 and maxCost = 0.05 then the trade requires a minimum of
78   %0.05*1000/100=0.5 dollar credit to enter the trade. If maxCost=-0.05 then
79   %it requires a maximum of 0.5 dollar debit to enter the trade.
80   maxCostRange = [-0.65 0.35];
81   %percentage points relative to margin required to exit a trade in profit
82   maxProfitRange = [0.0 1];
83   %percentage points relative to margin required to exit a trade in loss
84   maxLossRange = [0.0 1];
85   %how many days until expiration to exit a trade
86   exitDTERange = [1 30];
87   %how many days until expiration to enter a trade
88   dteRange = [35 100];
89
90   % dimension of problem
91   D   = 8;
92   % size of population
93   NP  = 25;
94   % differentiation constant
95   F   = 0.3;
96   % crossover constant
97   CR  = 0.5;
98   % number of generations to stop at
99   lagGen = 75;
100  %number of random restarts
101  runs = 10;
102
103  Boundary.mappingType = mappingType;
104  Options.mappingType = mappingType;
105  OptionsOS.mappingType = mappingType;
106
107  %function to get the month number from a date string.
108  get_month = @(dt) floor(mod((dt/100), 100));
109  start_range = 1;
110  lastMonth =  get_month(tradingDays(1));
111  currentMonth = get_month(tradingDays(1));
112  count = 1;
113  NUM_MONTHS = 138;
114  monthsDayRange = zeros(NUM_MONTHS,2);
115  n = 1;
116  %find the integer ranges from each moth based on the trading days.
117  while n < length(tradingDays)
```

```
    while currentMonth == lastMonth
     n = n + 1;
     currentMonth = get_month(tradingDays(n));
    end
    ender = n - 1;
    monthsDayRange(count, :) = [start_range ender];
    count = count + 1;
    lastMonth = currentMonth;
    start_range = n;
end

%Initialize DE generation parameters
fitnessProgress = {};
Pop = zeros(D,NP);  % population
Fit = zeros(2,NP);  % fitness of the population
isAllFitness = {};
osAllFitness = {};
osPopulation = [];
osDates = [];
osPfts = [];
osROC = [];
osReasons = [];
bestIndAll = [];
Boundary.llsRange = llsRange;
Boundary.ulsRange = ulsRange;
Boundary.ssRange = ssRange;
Boundary.maxCostRange = maxCostRange;
Boundary.maxProfitRange = maxProfitRange;
Boundary.maxLossRange = maxLossRange;
Boundary.exitDTERange = exitDTERange;
Boundary.dteRange = dteRange;
Boundary.inputRange = [0 1];

% index of the best solution in the population
iBest = 1;
% Set the number of in and out-of-sample trading months
isMaxMonths = 12;
osMaxMonths = 1;

Options.isOOS = false;
Options.monthsDayRange = monthsDayRange;
OptionsOS.isOOS = true;
OptionsOS.monthsDayRange = monthsDayRange;

%for all months optimize the BWB.
for n=1:NUM_MONTHS-isMaxMonths-osMaxMonths%
  bestInd = [];
  bestFit = -Inf;
  for iRun=1:runs
    fitnessProgress = [];
    %find where should the out of sample month should start
    osMonths = (n+isMaxMonths-1)+osMaxMonths;
    nPop = 0;
    %for some configurations it takes too long to find total population, in
    %that case I just want to exit and skip that month
    tries = 0;
    %start intelligent initialization
    while 1
      %generate random individual vector and decode it.
      Y = rand(D,1);
      [Trade, ~] = decode_individual(Y, Boundary);
      %trading day of end of range - maximum possible holding day
      Trade.tradingMonths = find_months_range(n, isMaxMonths, Trade.dte, Trade.exitDte,
monthsDayRange);
```

```
181         %if individual passes all constraints, add it to the initial
182         %population.
183         [fitness, ok, ~, ~, ~] = get_fitness(Trade, Options);
184         if ok > 0
185           nPop = nPop + 1;
186           Fit(1, nPop) = fitness;
187           Trade.tradingMonths = osMonths;
188           [fitness, ~, ~, ~, ~] = get_fitness(Trade, OptionsOS);
189           Fit(2, nPop) = fitness;
190           Pop(:, nPop) = Y;
191           % exit when we have enough individuals
192           if nPop == NP
193             break
194           end
195         end
196         tries = tries + 1;
197         % number of tries is proportional to the population
198         if tries > min([100000 7500*(nPop+1)])
199           break
200         end
201       end
202       if tries > min([100000 7500*(nPop+1)])
203         fprintf("Exceeded intelligent initalization tries\n")
204         continue
205       end
206       [~, iBest] = max(Fit(1,:));
207       lastFit = 1;
208       g = 1;
209       %stop searching the fitness space once the best individual does not
210       %improve
211       while lastFit > 0.001
212         % get random population members
213         rot = randperm(5);
214         r1 = randperm(NP);
215         r2 = circshift(r1, rot(1));
216         r3 = circshift(r2, rot(2));
217         pop1 = Pop(:, r1);
218         pop2 = Pop(:, r2);
219         pop3 = Pop(:, r3);
220
221         %perform mutation and crossover
222         crossLocations = rand(NP,D) < CR;
223         mutantPop = pop3 + F * (pop1 - pop2);
224         newPop = Pop.*(~crossLocations(1,:))' + mutantPop.*crossLocations(1, :)';
225         % for each individual check their fitness and if they belong in the
226         % new population
227         for iInd = 1:NP
228           originalX = Pop(:, iInd);
229           X = newPop(:, iInd);
230           X = enforce_boundaries(X, originalX);
231           [Trade, ~] = decode_individual(X, Boundary);
232           Trade.tradingMonths = find_months_range(n, isMaxMonths, Trade.dte, Trade.exitDte
    , monthsDayRange);
233           [f, ok, ~, ~, ~] = get_fitness(Trade, Options);
234
235           % if trial is better or equal then current
236           if (1/f) <= (1/Fit(1,iInd)) && ok > 0
237             Trade.tradingMonths = osMonths;
238             [f_oos, ~, ~, ~] = get_fitness(Trade, OptionsOS);
239             % replace current by trial
240             Pop(:,iInd) = X;
241             Fit(1,iInd) = f ;
242             Fit(2,iInd) = f_oos ;
243             % if trial is better then the best, update the best's index
```

```
244             if (1/f) < (1/Fit(1,iBest))
245               iBest = iInd ;
246             end
247           end
248         end
249         if mod(g, 39) == 0
250           fprintf("Run %d OS Month %d Generation %d best is fitness %f os fitness %f\n",
    iRun, osMonths, g, Fit(1,iBest), Fit(2,iBest));
251         end
252         fitnessProgress(g, 1) = Fit(1, iBest);
253         fitnessProgress(g, 2) = Fit(2, iBest);
254         %save the fitness to check if it changed later
255         if g > lagGen+1
256           lastFit = fitnessProgress(g, 1) - fitnessProgress(g-lagGen-1, 1);
257         end
258         g = g + 1;
259       end
260       %save the fitness for this in and out-of-sample period.
261       isAllFitness{osMonths}{iRun} = fitnessProgress(:, 1)';
262       osAllFitness{osMonths}{iRun} = fitnessProgress(:, 2)';
263       if Fit(1,iBest) > bestFit
264         bestInd = Pop(:,iBest);
265         bestFit = Fit(1,iBest);
266       end
267     end
268     if length(bestInd) == 0 && tries > min([100000 7500*(nPop+1)])
269       bestInd = zeros(D,1);
270     end
271
272     bestIndAll(:,osMonths) = bestInd;
273   end
274   save('results.mat', 'isAllFitness', 'osAllFitness', 'bestIndAll')
275
```

```
function [profit_out, reason, daily_returns, dates, Greeks] = bwb_manage_params(Trade,
Options)
   %{
  This function manages a trade by entering and exiting at some given
  specific conditions.

  Given a BWB trade parameters and its trade Options, manage the strategy.
  Returns the total profit, the reason it exited in coded format, the daily
  P/L and dates of the trade, and the daily delta and gamma greeks.
  %}
  global expirationDaysCell;
  global expirationDays;
  global tradingDays;
  global NORM_MAPPING;
  global DELTA_MAPPING;
  global POINTS_MAPPING;

  profit_out = eps;
  reason = -1;

  REASON_MAX_LOSS = 0;
  REASON_MAX_PROFIT = 1;
  REASON_EXIT_DTE = 2;
  REASON_EXPIRATION = 3;

  Greeks.delta = [];
  Greeks.gamma = [];

  chains = Trade.chains;
  %get all chains with the required expiration date
  exp_dates = expirationDaysCell{expirationDays == chains(1, 2)};

  %get each strike's daily data
  strks = exp_dates(:, 8);
  llp = exp_dates(strks == chains(1, 8),:);
  sp = exp_dates(strks == chains(2, 8),:);
  ulp = exp_dates(strks == chains(3, 8),:);

  %if any of the legs does not have data or if any of them has less data
  %than the others, we will not trade (or trade on incomplete data).
  if isempty(ulp) || isempty(sp) || isempty(llp) || ...
        size(llp, 1) ~= size(sp, 1) || size(llp, 1) ~= size(ulp, 1)
      daily_returns = [];
      dates = [];
      return;
  end

  %get the premium of the entry date. 9 is bid, 10 is ask, 15 is number
  %of contracts.
  premium_in = -((chains(1, 9) * 1/3) + chains(1, 10) * 2/3) * chains(1, 15) + ...
                ((chains(2, 9) * 2/3) + chains(2, 10) * 1/3) * chains(2, 15) + ...
               -((chains(3, 9) * 1/3) + chains(3, 10) * 2/3) * chains(3, 15);

  %Get the row number locations in the global chain.
  exp_locs = find(llp(:, 1) == chains(1, 1));
  % Get the number of days for trading.
  days = size(llp, 1);

  %Initialize variables
  daily_returns = zeros(days + 1, 1);
  dates = zeros(days + 1, 1);
  Greeks.delta = zeros(days + 1, 1);
  Greeks.gamma = zeros(days + 1, 1);
  daily_returns(1) = 0;
```

```
64      dates(1) = chains(1, 1);
65      Greeks.delta(1) = chains(1, 11) - chains(2, 11) * 2 + chains(3, 11);
66
67      %Calculating gamma value is cpu intensive. Comment this code to improve
68      %performance.
69      [~, llgamma, ~] = getGreeks(chains(1, :));
70      [~, sgamma, ~] = getGreeks(chains(2, :));
71      [~, ulgamma, ~] = getGreeks(chains(3, :));
72      Greeks.gamma(1) = llgamma - sgamma * 2 + ulgamma;
73
74      %start on the first trading day.
75      iDay = exp_locs(end)+1;
76      %While there are trading days available and we have not forcibly exited,
77      %continue trading
78      while iDay <= days && reason < 0
79        %Get the current premium to exit the trade.
80        premium_out =  ((llp(iDay, 9) * 2/3) + llp(iDay, 10) * 1/3) * chains(1, 15) + ...
81                      -((sp(iDay, 9) * 1/3) + sp(iDay, 10) * 2/3) * chains(2, 15) + ...
82                       ((ulp(iDay, 9) * 2/3) + ulp(iDay, 10) * 1/3) * chains(3, 15);
83        %Calculate the current return and store it.
84        day_return = premium_out + premium_in ;
85        daily_returns(iDay + 1 - exp_locs(end)) = day_return;
86        dates(iDay + 1 - exp_locs(end)) = llp(iDay, 1);
87
88        %Delta is calculated from the option chain.
89        Greeks.delta(iDay + 1 - exp_locs(end)) = llp(iDay, 11) + sp(iDay, 11) * -2 + ulp(
    iDay, 11);
90        [~, llgamma, ~] = getGreeks(llp(iDay, :));
91        [~, sgamma, ~] = getGreeks(sp(iDay, :));
92        [~, ulgamma, ~] = getGreeks(ulp(iDay, :));
93        Greeks.gamma(iDay + 1 - exp_locs(end)) = llgamma - sgamma * 2 + ulgamma;
94
95        %check the exit conditions.
96        if day_return < -Trade.maxLoss * (Trade.margin/100)
97            reason = REASON_MAX_LOSS;
98        elseif day_return > Trade.maxProfit * (Trade.maxWin/100)
99            reason = REASON_MAX_PROFIT;
100       elseif llp(iDay,3) <= Trade.exitDte
101           reason = REASON_EXIT_DTE;
102       elseif llp(iDay,1) >= tradingDays(end)
103           reason = REASON_EXPIRATION;
104       end
105
106       iDay = iDay + 1;
107     end
108
109     %Only keep the days that actually were traded.
110     profit_out = day_return;
111     daily_returns = daily_returns(1:iDay - exp_locs(end));
112     dates = dates(1:iDay - exp_locs(end));
113     Greeks.delta = Greeks.delta(1:iDay - exp_locs(end));
114     Greeks.gamma = Greeks.gamma(1:iDay - exp_locs(end));
115   end
116
```

```
function [maxLoss, isUndefined, maxWin] = check_max_loss(strks)
  %{
  This function returns the maximum loss and maximum profit a BWB can obtain.
  It also checks if the BWB is risk defined.
  %}
  maxPrice = 300;
  minPrice = 70;
  isUndefined = false;
  maxLoss = -1;
  %create large enough range of prices to check to get complete picture of
  %P/L graph and know if there is undefined loss
  rangeOfPrices = [minPrice; strks(:,1); maxPrice];
  pl = zeros(length(rangeOfPrices),1);
  %for each price, get the expiration return at that point
  K = strks(:,1);
  value = strks(:,2);
  nContracts = strks(:,4);
  for n=1:length(rangeOfPrices)
    s = rangeOfPrices(n);
    otmLocs = s >= K;
    itmLocs = ~otmLocs;
    otm = -value .* nContracts;
    itm = (-sign(value).*(s - K) - value) .* nContracts;
    pl(n)=  sum(round(otm .* otmLocs,3) + round(itm .* itmLocs,3));
  end
  %get p/l
  maxLoss = abs(min(pl)*100);
  maxWin = max(pl)*100;
  isLeftUndefined = false;
  isRightUndefined = false;
  %if the leftmost or rightmost price is decreasing it means it is undefined
  %single is needed to truncate the precision.
  if single(pl(2)) > single(pl(1))
    isLeftUndefined = true;
  end
  if single(pl(length(rangeOfPrices)-1)) > single(pl(end))
    isRightUndefined = true;
  end
  isUndefined = isLeftUndefined || isRightUndefined;
  if ~isUndefined
    %if the rightmost price is less or equal than the leftmost price, we
    %do not have a true BWB
    if single(pl(1)) >= single(pl(end))
      isUndefined = true ;
    end
  end
end
```

```
1  function x1 = decode(x, output_range)
2    %{
3    This function maps a [0, 1] variable to the range given.
4    %}
5    x1 = output_range(1) + (output_range(2) - output_range(1)) * x;
6  end
```

```
function [Trade, X] = decode_individual(Y, Boundary)
  %{
  Transforms a population vector in the range [0, 1] to a Trade structure
  and a trade vector.
  %}
  Trade.ulStrike = decode(Y(2), Boundary.ulsRange);
  Trade.sStrike = decode(Y(3), Boundary.ssRange);
  Trade.llStrike = decode(Y(1), Boundary.llsRange);
  Trade.maxCost = decode(Y(4), Boundary.maxCostRange);
  Trade.maxProfit = decode(Y(5), Boundary.maxProfitRange);
  Trade.maxLoss = decode(Y(6), Boundary.maxLossRange);
  Trade.exitDte = decode(Y(7), Boundary.exitDTERange);
  Trade.dte = decode(Y(8), Boundary.dteRange);
  Trade.lots = 1;

  X(1) = Trade.llStrike;
  X(2) = Trade.ulStrike;
  X(3) = Trade.sStrike;
  X(4) = Trade.maxCost;
  X(5) = Trade.maxProfit;
  X(6) = Trade.maxLoss;
  X(7) = Trade.exitDte;
  X(8) = Trade.dte;
end
```

```
function X = enforce_boundaries(X, X0)
  %{
  Enforces the boundaries of an individual vector.
  If a dimension is out of bounds, use the original dimension to calculate
  a new value.
  %}
  function y = bound(y, y0, rg)
    if y > rg(2)
      y = rg(2) + rand*(y0 - rg(2));
    elseif y < rg(1)
      y = rg(1) + rand*(y0 - rg(1));
    end
  end

  X(1) = bound(X(1), X0(1), [0 1]);
  X(2) = bound(X(2), X0(2), [0 1]);
  X(3) = bound(X(3), X0(3), [0 1]);
  X(4) = bound(X(4), X0(4), [0 1]);
  X(5) = bound(X(5), X0(5), [0 1]);
  X(6) = bound(X(6), X0(6), [0 1]);
  X(7) = bound(X(7), X0(7), [0 1]);
  X(8) = bound(X(8), X0(8), [0 1]);
end
```

```
function [ulc, sc, llc, total, marginUsed, isUndefined, maxWin] = find_chains_params(
Trade, Options)
  %{
  Given the entry parameters, find the option chains that satisfy them.
  %}
  global NORM_MAPPING;
  global DELTA_MAPPING;
  global POINTS_MAPPING;

  chain = Trade.chain;
  %variable to take into account that put deltas are negative.
  delta_const = 1;

  %get the closest strike based on the mapping
  if Options.mappingType == POINTS_MAPPING
    %close price is ATM
    [~, I] = min(abs((chain(:, 8) - (Trade.close + Trade.ulStrike))));
    upperLongStrike = chain(I, 8);
    map_loc = 8;
    %forcing short strike to be OTM
    shortStrike = (Trade.close + Trade.ulStrike) - Trade.sStrike;
  elseif Options.mappingType == NORM_MAPPING
    %1.0 is ATM
    [~, I] = min(abs((chain(:, 14) - (1.0 + Trade.ulStrike))));
    upperLongStrike = chain(I, 14);
    map_loc = 14;
    %forcing short strike to be OTM
    shortStrike = (1.0 + Trade.ulStrike) - Trade.sStrike;
  elseif Options.mappingType == DELTA_MAPPING
    %-0.5 delta is ATM, -1.0 < x < -0.5 is ITM, -0.5 < x < 0.0 is OTM
    %since delta is negative, and ulStrike is positive (to simulate how much above)
    %have to subsctract ulStrike from ATM to go ITM
    [~, I] = min(abs((chain(:, 11) - (-0.5 - Trade.ulStrike))));
    upperLongStrike = chain(I, 11);
    map_loc = 11;
    delta_const = -1;
    %forcing short strike to be OTM
    shortStrike = (-0.5 - Trade.ulStrike) + Trade.sStrike;
  end

  lowerLongStrike = shortStrike - Trade.llStrike * delta_const;

  %find the option chains closes to the strikes selected
  [i, I] = min(abs((chain(:, map_loc) - upperLongStrike)));
  [j, J] = min(abs((chain(:, map_loc) - shortStrike)));
  [k, K] = min(abs((chain(:, map_loc) - lowerLongStrike)));
  ulc = chain(I, :);
  sc = chain(J, :);
  llc = chain(K, :);

  %calculate how much will cost to get into trade for 1 lots.
  %9 is bid, 10 is ask,
  credits = ((sc(9)   * 2/3) + sc(10) * 1/3) * 2;
  debit_upper = ((ulc(10) * 2/3) + ulc(9) * 1/3);
  debit_lower = ((llc(10) * 2/3) + llc(9) * 1/3);
  debits = debit_upper + debit_lower;
  total = credits - debits;

  %set strks structure
  strks = [llc(8)   ((llc(9) * 1/3) + llc(10) * 2/3) 0  1*Trade.lots;
           sc(8)  -((sc(9) * 2/3) + sc(10) * 1/3)   0  2*Trade.lots;
           ulc(8)   ((ulc(9) * 1/3) + ulc(10) * 2/3) 0  1*Trade.lots;];
  %check if resulting bwb is risk defined as well.
  [marginUsed, isUndefined, maxWin] =  check_max_loss(strks);
```

```
64
65    %if costs more to get in than allowed, if any cost is 0 (means there must
66    %be no volume), the difference in strikes is too big (most likely went to
67    %the edges of the chain), or the bid of the lower leg is less than 0.1
68    %(most likely there is no volume and therefore would not be able to get
69    %in into the trade) then fail to get into the trade
70    if total < (marginUsed*Trade.maxCost)/100 || credits == 0 || debit_upper == 0 ||
      debit_lower == 0 || ...
71            abs(i) > Options.minThresh || abs(j) > Options.minThresh || ...
72            abs(k) > Options.minThresh || llc(9) <= 0.08 || isUndefined || ...
73             ulc(8) == sc(8) || llc(11) < -0.5 || ulc(11) < -0.75
74      ulc = [];
75      sc = [];
76      llc = [];
77    end
78  end
```

```m
function rg = find_months_range(startMonth, numberOfMonths, dte, exitDTE, monthRanges)
  %{
  finds the maximum number of months that on a worse case
  scenario, would stay within the DTE range.
  EX: if I want only 24 months of range, but my worst case scenario for
  staying in a position is 40 days (dte - exitDTE) then this tool finds
  the day number and matches to the monthsRange
  number of months is how many total months needed from startMonth
  %}
  endDay = floor(monthRanges(startMonth + numberOfMonths - 1, 2) - (dte - exitDTE));
  monthLoc = (monthRanges(:, 1) <= endDay) & (endDay <= monthRanges(:, 2));
  rg = (startMonth:1:find(monthLoc));
end
```

```m
function [fitness, ok, Returns, prob_profit, Stats] = get_fitness(Trade, Options)
  %{
  Calculates the fitness of the Trade given its returns.
  %}
  ok = 0;
  %get historical profits
  [Returns, Stats]  = get_hist_bwb_profits(Trade, Options);
  %if there are not trades or it traded for less than 2 days total, it is
  %not valid.
  if isempty(Returns.pfts) || length(Returns.dailyPortfolio) <= 2
    ok = -3;
    prob_profit =0;
    fitness = eps;
    return
  end
  %get the returns statistics.
  [cum_returns, ~, max_drawdown, ~] = ...
              get_returns_stats_single(Returns.dailyPortfolio,  Options);
  %calculate the fitness and probability of profit
  fitness = cum_returns(end) - max_drawdown/3;
  prob_profit = sum(Returns.pfts>0) / length(Returns.pfts);
  %if the trade satisfies the requirements or it is an out-of-sample trade,
  %it is correct.
  if Options.isOOS || fitness >= 0 && prob_profit>=0.7
      ok = 1;
  end
end
```

```m
function [Returns, Stats] = get_hist_bwb_profits(Trade, Options)
  %{
  finds the daily returns of a BWB trade given its entry and exit options.

  %}
  global tradingDays;
  global tradingDaysCell;

  Returns.pfts = [];
  Returns.pftsDay = [];
  Stats.cP = [];
  Returns.dailyPortfolio = zeros(size(tradingDays));
  Returns.dates = zeros(size(tradingDays));
  Stats.margins = [];
  Stats.reasons = [];
  Stats.llcs = {};
  Stats.scs = {};
  Stats.ulcs = {};
  Stats.totals = [];
  Returns.roc = [];
  totalTrades = 0;

  %for all months I want to trade
  for iMonth = 1:length(Trade.tradingMonths)
    %get the start and end day of that month (its range)
    rg = Options.monthsDayRange(Trade.tradingMonths(iMonth), :);
    jDay = rg(1);
    first_day = jDay;
    %trade for all days withing the range
    while jDay < rg(2)
      %if it is an out-of-sample trade, make sure we exit if no trades are
      %found half of the trading month
      if Options.isOOS
        if jDay >= first_day + 9
          break
        end
      end

      %get the current's day option chains and get the one with the closest
      %required DTE
      todays_chain = tradingDaysCell{jDay};
      [~, k] = min(abs(todays_chain(:,3) - Trade.dte)); %find closest DTE
      todays_chain = todays_chain(todays_chain(:,2)==todays_chain(k,2),:);
      closePrice = todays_chain(1,13);
      Trade.close = closePrice;
      Trade.chain = todays_chain;

      %find the strike chains of the BWB in that DTE chain
      Options.jDay = jDay;
      [ulc, sc, llc, total, marginUsed, ~, maxWin] = find_chains_params(Trade, Options);

      %if it didnt find a suitable trade, go to the next day
      if isempty(ulc)
        jDay = jDay + 1;
        continue;
      end

      Trade.chains = [llc Trade.lots;
                      sc  2 * Trade.lots;
                      ulc Trade.lots];
      Trade.margin = marginUsed;
      Trade.maxWin = maxWin;
      %get the returns of trading today until the exit conditions are met
      [profit_out, reason, dailyReturns, dates, Greeks] = bwb_manage_params(Trade,
```

```
    Options);
65
66        %if for any reason the trading fails, go to next day
67        if reason < 0
68          jDay = jDay + 1;
69          continue;
70        end
71
72        %store the returns and trade information for later use.
73        totalTrades = totalTrades + 1;
74        Returns.pfts(totalTrades) = profit_out;
75        Returns.entryDates(totalTrades) = todays_chain(1,1);
76        Returns.roc(totalTrades) = profit_out/(marginUsed/100);
77        Returns.Greeks = Greeks;
78        dailyReturns = diff(dailyReturns);
79        for d = 1:length(dailyReturns)
80          Returns.dailyPortfolio(jDay+d) = Returns.dailyPortfolio(jDay+d) + dailyReturns(d
    )*100 + eps;
81          Returns.dates(jDay+d) = dates(d);
82        end
83        Stats.cP(totalTrades) = closePrice;
84        Stats.margins(totalTrades) = marginUsed;
85        Stats.reasons(totalTrades) = reason;
86        Stats.totals(totalTrades) = total;
87        Stats.llcs{totalTrades} = llc;
88        Stats.scs{totalTrades} = sc;
89        Stats.ulcs{totalTrades} = ulc;
90
91        %if it is an out-of-sample trade, we only trade once.
92        if Options.isOOS
93          break;
94        else
95          jDay = jDay + 1;
96        end
97      end
98    end
99    %only get the days traded.
100   temp_portoflio = Returns.dailyPortfolio;
101   trade_start = find(temp_portoflio, 1, 'first');
102   temp_portoflio = flip(temp_portoflio);
103   trade_end = length(temp_portoflio) - find(temp_portoflio, 1, 'first') + 1;
104   Returns.dailyPortfolio = Returns.dailyPortfolio(trade_start:trade_end);
105   Returns.dates = Returns.dates(trade_start:trade_end);
106 end
```

```m
function [sharpe, sortino, movReturns, movVolatility] = get_historic_metrics(
nonCumReturns, days)
  %{
  calculates the daily returns' metrics.
  %}
  movVolatility = movstd(nonCumReturns, days, 'Endpoints','discard');
  movReturns = movmean(nonCumReturns, days, 'Endpoints','discard');
  sharpe = movReturns./movVolatility;

  cum_returns = nonCumReturns;
  cum_returns(cum_returns>=0)=0;
  DD = cum_returns.^2;
  movDD = sqrt(movmean(DD, days, 'Endpoints', 'discard'));
  sortino = movReturns./movDD * sqrt(252);

end
```

```
function [cum_returns, annual_vol, max_drawdown, annual_returns] =
get_returns_stats_single(returns, Options)
  %{
  calculates the statistics of the equity curve.
  %}

  %returns are raw dollar values, non commulative
  initialPortfolio = 10000;
  %daily annual volatility
  ann_factor  = 252;
  rets = initialPortfolio + cumsum(returns);
  rets_pct =  1 + (diff(rets)./rets(1:end - 1, :));

  cum_returns = cumprod(rets_pct)  - 1;

  retsStd = std(rets_pct);
  annual_vol = retsStd .* sqrt(ann_factor);

  % max drawdown
  dd = zeros(length(rets) - 1, 3);
  for k = 1:length(rets)-1
     [pmax, i] = max(rets(1: k + 1));
     [pmin, j] = min(rets(i: k + 1));

     dd(k,:) = [(pmax - pmin), i, i + j - 1];
  end
  [maxdd, k] = max(dd(:, 1));
  mx = rets(dd(k, 2)); % dd(k,2) is the index where the max occurs
  mn = rets(dd(k, 3)); % dd(k,3) is the index where the min occurs
  max_drawdown = maxdd/mx;

  annual_returns = (1 + cum_returns(end)) ^ (1 / 1/12) - 1;
end
```

```
function [delta, gamma, price] = getGreeks(chain)
  %{
  This function calculates the theoretical greeks and price of an option
  chain that has American style expiration.
  This function wraps around Matlab's optstocksensbybjs function.
  %}
  asset_price = chain(13);
  settlement_date = datestr(datenum(num2str(chain(1)), 'yyyymmdd'),'mmm-dd-yyyy');
  maturity_date = datestr(datenum(num2str(chain(2)), 'yyyymmdd'),'mmm-dd-yyyy');
  strike = chain(8);
  risk_free_rate = 0.02;
  volatility  = chain(12);
  dividend_amount = 0.01;
  if volatility < 0
    delta = eps;
    gamma = eps;
    price = eps;
  else

    spec = stockspec(volatility, asset_price, {'continuous'}, dividend_amount);
    risk_free_rate_spec = intenvset('ValuationDate', settlement_date, 'StartDates',
settlement_date,...
                                    'EndDates', maturity_date, 'Rates', ...
                                    risk_free_rate, 'Compounding', -1, 'Basis', 1);

    OptSpec = {'put'};
    OutSpec = {'Delta', 'Gamma', 'Price'};

    [delta, gamma, price] = optstocksensbybjs(risk_free_rate_spec, spec, ...
                                              settlement_date, maturity_date,...
                                              OptSpec, strike, 'OutSpec', OutSpec);
  end
end
```

```
function x = pad(d)
  %{
  pads a 0 if necessary to an integer. Used for date parsing.
  %}
  if d < 10
    x = strcat('0',num2str(d));
  else
    x = num2str(d);
  end
end
```