Numerical experiments on resistance optimal strategy - Part II

Camilo Andrés García Trillos - `camilo.garcia@mathmods.eu`
MathMods - Mathematical Modeling in Engineering
Industrial Seminar

Advisor: Etienne Tanré

July 2009

Abstract

Although its prediction power is still a controversial subject, technical analysis is frequently used by traders around the globe to support some of their investment decisions. The term refers to a set of tools, graphic and numeric indicators, that intend to predict the future behavior of an asset based on its historical values.

One of the concepts that appears frequently in technical analysis is resistance. We will refer to resistance level as a given price level when market pressures seem to force a repeated maximum. Although the study of some historical data shows the eventual appearance of resistance levels, most of the mathematical models used for stock prices, including the well known Black and Scholes model, do not contemplate this kind of phenomenon.

In [3], Bérard Bergery et al. developed a mathematical model derived from the Black and Scholes model for including resistance phenomena. Likewise, the article presented a trading strategy for optimizing the logarithm of the returns when considering a portfolio with a risky and a risk-less asset assuming the former follows the resistance mathematical model.

Naturally it arises the question whether this strategy may be used with good results in a practical environment. In our work, we perform several tests to determine this possibility. We simulate stochastic paths following the model for resistance presence, for comparing via Monte Carlo methods the wealths obtained by applying the optimal strategy for a classical Black and Scholes model, a simple technical analysis strategy (based on moving average and resistance selling), and the optimal strategy for the resistance model.

The comparisons are made under full knowledge of the parameters and probability laws involved, as well as in different scenarios for representing estimation errors.

Interestingly, the tests showed the optimal strategy with resistance proved to create greater wealth in our tests than the other strategies as long as the estimated law for the number of down-crossings (i.e. the number of times the price travels from the resistance level to a given fixed level) is close enough to the real one. The determination of this law appears therefore as a critical component before using the strategy in real price data.

However, we failed to find real series that show resistance with enough persistence to allow room for investment prediction.

Contents

Introduction to the second part

A natural question that follows the development of a mathematical strategy is the possibility to implement it for its use in industry. Indeed, it is not straightforward that a strategy, even an optimal one, would give good results when used in a real context.

On one hand, there is a question about the implementation of the strategy itself. Frequently, mathematical based strategies depend on parameters (such as the drift or the volatility) that are generally unknown for the traders. In the particular case of the optimal resistance strategy, it depends on the knowledge of at least five parameters and a probability distribution. Therefore parameter estimation techniques must be used, with the restriction that they need to be simultaneously fast and relatively easy to implement for them to allow its use and understanding by non mathematicians, yet precise enough for the strategy to deliver still interesting returns.

On the other hand, it remains the question about the model error. Mathematical strategies are constructed under assumptions that the real financial prices may not follow. In our case of concern, the model assume all the hypothesis of the Black and Scholes model plus those that allow the presence of a resistance.

This is the main goal of the work performed during our internship in INRIA. The present document corresponds to the second part of this work. The first part presents the model used and the algorithms for simulating stochastic paths following this model. This concepts will be assumed for this part.

The present document is divided in two main chapters. In Chapter 1 we will study the implementation of the strategy itself, while in Chapter 2 we will focus on the possible model error.

Chapter 1

Implementation of the strategy

A financial series strategy may be described as the investment decisions a trader should undertake as a function of time.

In order to use the strategy formula derived in [3], we must implement it, meaning transforming it in a valid and usable algorithm for deciding the investment quantities. As shown in the first part, in our particular case of interest, the mathematical form of the strategy is a fractional function with some infinite sums that may be truncated in most cases, becoming an easily implementable function.

However, some of the variables involved in this calculation are unknown. Therefore, in this case implementing the strategy includes designing an algorithm for detecting the resistance level that the financial series show (section 1.1), and estimating the model parameters (section 1.2) that best fit the series to the mathematical model. We will focus our attention in this report in those two steps.

The goodness of the output of the implementation must be tested. We will use Monte Carlo technique to approximate the probability distribution of the wealth generation of the implemented strategy, which implies simulating several different paths for evaluating the output (Section 1.3).

1.1 Resistance detection

A resistance level is commonly defined as *"The price at which a stock or market can trade, but not exceed, for a certain period of time."* [1]. Although this definition appears clear, it does not clarify how to identify a resistance level.

In technical analysis, there is a very wide range of methodologies to find a resistance line. For example, in [4], the resistance and support levels are defined

[1] *www.investopedia.com*, June 2009

in terms of channels constructed around price charts local maxima and minima, while in [5] some methods called Fibonacci studies (based on using aural proportion ratios) are used for this purpose.

Although we are in principle not interested in solving the problem of finding a resistance level by itself, any attempt to test by Monte Carlo means the benefits of using an implemented version of the optimal resistance strategy implies developing an algorithm to identify a resistance level.

Consider a price following the Black and Scholes model. The probability that the price will have two local maxima with the same value is 0. We propose an algorithm based on this property and on the development of the model itself, and we will test it against paths simulated following the resistance model.

Define the time window $W = [t_0, t_N]$ as the time frame in which the detection will be performed. An initial idea is that for a given stochastic path Z_t a resistance is detected if for an integer constant $\eta > 2$ and a given real "small" constant S_0^+ we can find η local maxima of the path in W, $M = \{ZM_1, \ldots, ZM_\eta\}$ with corresponding times $T = \{TM_1, \ldots, TM_{ZM}\}$, such that the maxima are pairwise closer than S_0^+ and M contains the maximum point in the interval $[\min(T), t_f]$.

We test this idea using Monte Carlo methods, when our parameters are known. We test it with $\eta = 2, 3$. While working with simulated data, we will assume a value for $S_0^+ := (\exp(\sigma \cdot \epsilon) - 1) * S_0$ since in this way the parameter coincides with the gap given over the resistance level, assuring maximum detection rate. The parameters used for the simulated paths are presented in Table 1.1

μ	σ	α	ϵ	Resistance Price
0.925	0.15	0.5	0.001	100

Table 1.1: Resistance test simulation parameters

The obtained results are shown in Table 1.2. Although this algorithm has a good detection rate, defined as the number of found resistance vs. number of trials error, it has a poor right detection error, defined as the percentage of times the simulated resistance level was found.

	$\eta = 2$	$\eta = 3$	$\eta = 4$
Detection Rate	100%	100%	100%
Right detection rate	4.474%	3.802%	5.2381%
Avg. resistance detected	92.62%	93.46	94.14

Table 1.2: Results for the first detection resistance algorithm

The reason behind the false resistance detection is that the algorithm may identify small variations or noise around a fixed value as a resistance level. We are not interested in this kind of resistance detection since it gives no space for defining and executing a trading strategy. Therefore, we will introduce one extra parameter, γ, that asks the path to descend a sufficient amount between local maxima to account for the trading space, i.e., we apply the definition given replacing the set M for the set $M' = \{ZM'_1, \ldots, ZM'_\eta\}$ of local maxima such that $\forall i \exists t_i \in [TM'_i, TM'_{i+1}]$ such that $|Z_{t_i} - ZM'_i| < \gamma$

In order to choose an appropriate value of γ for our tests with simulated data, we may define it as a function of α as $\gamma := (1 - \exp(-\alpha \cdot \sigma)) * S_0$. The Table 1.3 shows the result of applying the method with the given parameters. The Figure 1.1 shows a detection example, as well as indication lines for the parameters of detection.

	$\eta = 2$	$\eta = 3$	$\eta = 4$
Detection Rate	99.759%	88.75 %	79.798%
Right detection rate	71.256%	99.5475%	100%
Avg. resistance detected	98.66	100.133%	100.173%

Table 1.3: Results for the detection resistance algorithm. The final algorithm performs much better with respect to false resistances.

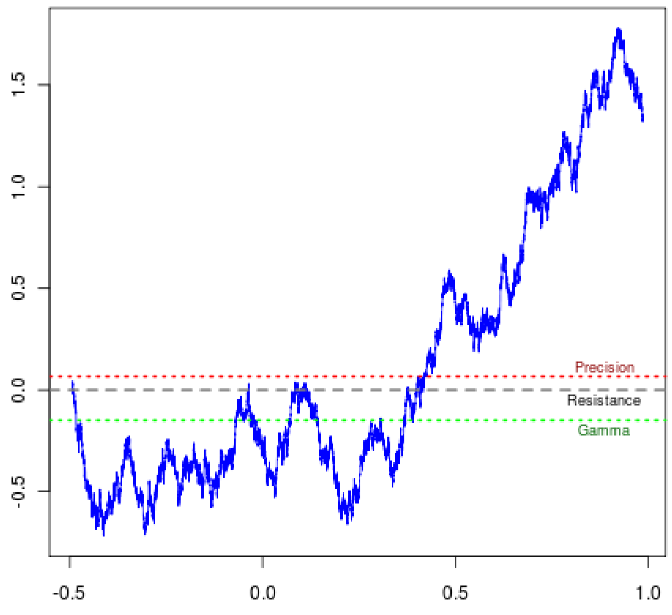

Figure 1.1: An example of resistance detection. The detected value and the two band levels S_0^+ (precision) and γ (gamma) are shown

The obtained results show that as long as nice values are used for the variables η,

S_0^+ and γ the algorithm gives both good resistance detection and right detection rates. A good compromise is choosing the number of touches as two or three. On section 2.1 we will deal with the problem of choosing this parameters when they are unknown for real series.

1.2 Parameter Estimation

The optimal resistance strategy developed in [3] is based on the knowledge of the parameters that characterize the underlying mathematical model. Since in general this parameters are not known, we need definitions and estimations to implement the strategy.

The problem of estimating parameters to adjust a financial series to a given mathematical modeling appears frequently in trading related activities when mathematical strategies are involved. A typical example is the volatility estimation techniques. The Figure 1.2 shows a measure of volatility of percentage daily changes for five years U.S. Treasury Notes. This volatility rates are used, for example, as risk indicators, and are obtained assuming normal random changes as the ones presented in the Brownian motion.

One interesting thing to note in Figure 1.2 is that the estimated value changes abruptly from one month to the next one. For example, between May and June 1988, the volatility changed by a factor of three from one month to the next one, while on September 1998 volatility changed by a factor of four in less than two months.

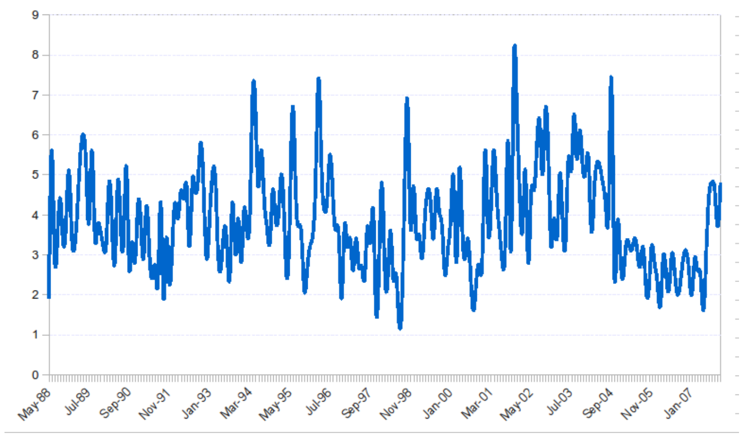

Figure 1.2: Volatility of 5yrs U.S. Treasury Notes between 1988 and 2007, measured as annualized standard deviation of the percentage change in daily price. Data Source: Chicago Board of Trade

This is just an example that shows that parameters are constantly changing. This is inconvenient for a mathematical model as the one we are dealing with, since the model was defined assuming constant parameters. Thus, we have two opposed situations: parameters are changing and if we assume that they are constant for a long time we will incur in considerable errors. On the other hand, the model is useless for changing parameters.

A compromise between the two competing constraints is using estimation time windows with a fixed length large enough for allowing nice parameter estimation and not too much such that parameter change in the series are reflected in the model. We lack of a standard procedure to follow to define this number, so it has to be proposed and tested individually depending on the results and the available information.

Let us assume that we already count with an estimation of the resistance level ($\hat{S}_0$) and the value of the upper band $\hat{S}_0^+$, and that we wish to perform parameter estimation in the time interval $[t_0, t_N]$. We need to derive results for estimating ϵ, σ, α and μ. Let us first consider the estimation of ϵ. This variable has the same role for the underlying Brownian motion with drift that the one S_0^+ plays for the price series. Therefore, following [3], it is convenient to define it as

$$\hat{\epsilon} := \frac{1}{\hat{\sigma}} \log(\hat{S}_0^+ / \hat{S}_0) \tag{1.1}$$

This lead us to the problem of obtaining a value for $\hat{\sigma}$. Recall from Part I of this work that if we define $X_t = \frac{1}{\sigma} \log(Z_t/S_0)$, then X_t satisfies the couple of stochastic differential equation:

$$dX_t = \begin{cases} \mu dt + d\tilde{B}_t & \text{for \textit{downcrossing} or \textit{free} phases} \\ -\mu \coth(\mu(\epsilon - X_t))dt + d\tilde{B}_t & \text{otherwise} \end{cases} \tag{1.2}$$

Where $\tilde{B}_t$ is a standard Brownian motion. Note that the only random part of this equation is due to the presence of $\tilde{B}_t$. Moreover we know that $\Delta\tilde{B}_{t_i} = \tilde{B}_{t_i} - \tilde{B}_{t_{i-1}}$ follows a normal distribution with mean 0 and variance $\Delta t_i = t_i - t_{i-1}$.

With this in mind, and defining $\Delta Y_{t_i} = \sigma \Delta X_{t_i} = \sigma(X_{t_i} - X_{t_{i-1}})$ we propose the following lemma. Its is a consequence of a similar lemma for independent identically distributed variables.

Lemma 1. *$\hat{\sigma} := \sqrt{\frac{1}{N-1} \sum_{i=0}^{N} (Y_{t_i} - \bar{y})^2}$, is an unbiased estimator of σ, with $\bar{y} = \frac{1}{N} \sum_{i=0}^{N} Y_{t_i}$.*

Similarly we would like to use a simple estimator for μ. Indeed, for the up-crossing and free phases, (1.2) allows to estimate μ using the sample mean of ΔX_t. However, the dependence of dX_t on X_t itself is an obstacle to apply this kind of solution. Moreover, since we do not know the value of α, we cannot separate the up-crossing and down-crossing phases.

A simultaneous estimation of $\hat{\mu}$ and $\hat{\alpha}$ appears therefore as the best solution. We can then define $\hat{\mu}$ and $\hat{\alpha}$ as the discretized Least Square Estimators, i.e. define them as the pair that solves

$$\min_{\sigma,\alpha} \left(\sum_{i=0}^{N} R_i^2 \right)$$

where

$$R_i = \begin{cases} \Delta X_t - \mu\Delta t & \text{for } \textit{downcrossing} \text{ or } \textit{free} \text{ phases} \\ \Delta X_t + \mu\coth(\mu(\epsilon - X_t))dt & \text{otherwise} \end{cases}$$

Note that the discretization approximation is performed using an implicit scheme. In any case the discretization error may diminish the goodness of the estimation, and therefore it is convenient to test the results obtained with this methodology. For implementing the minimization algorithm, we chose to perform the minimization in two steps: traversing first the α using a dichotomous search and then solving the gradient for μ.

As shown in Table 1.4 the results are not very satisfactory. The discretization errors strongly bias the estimation, particularly the one referring to μ. It is therefore better to use less information in order to gain accuracy: we will modify slightly the LSE estimation for using just the up-crossing phase, i.e. we will define $\hat{\mu}$ as the solution to

$$\min_{\sigma,\alpha}\left(\sum_{i=0}^{N} R_i^{'2}\right)$$

where

$$R_i^{'} = \begin{cases} \Delta X_t - \mu_\alpha\Delta t & \text{for } \textit{downcrossing} \text{ or } \textit{free} \text{ phases} \\ \Delta X_t + \mu_\alpha\coth(\mu(\epsilon - X_t))dt & \text{otherwise} \end{cases}$$

and

$$\mu_\alpha = \frac{1}{N'}\sum(\Delta X_{t_{i'}}/\Delta t_{i'})$$

	Estimated	Simulated	Bias	Relative Bias
Sigma	0.15 $\pm 10^{-4}$	0.15	0	0 %
Alpha	0.7197 $\pm$ 0.02	0.5	0.22	43.94%
Mu	2.4674 $\pm$ 0.12	0.93	1.54	166.94%

Table 1.4: Results when applying the first estimation method. The error are calculated as twice the standard deviation over the square root of the number of simulations.

Consequently, we will define $\hat{\mu} = \mu_{\hat{\alpha}}$. The propose modification proves to be an improvement with respect to the previous results with respect to μ as shown in Table 1.5 . However, as can be deduced from Figure 1.3, there is still an important dispersion around the values to estimate. Therefore, it is convenient to check the sensibility of the results obtained using the optimal strategy with respect to this parameters. This will be performed in Section .

Finally, we must remark that in addition to the estimation of the unknown parameters, it is necessary to find the probability law followed by the number of dowcrossings. This issue will be discussed on section 2.1.

	Estimated	Simulated	Bias	Relative Bias
Sigma	0.15 $\pm 10^{-4}$	0.15	$-2 \cdot 10^{-4}$	-0.16 %
Alpha	0.8680 $\pm$ 0.02	0.5	0.37	73.61%
Mu	0.7792 $\pm$ 0.15	0.93	-0.15	-15.76%

Table 1.5: Results when applying the modified estimation method. The error are calculated as twice the standard deviation over the square root of the number of simulations.

1.3 Simulated Data Tests

In the previous sections, we have setup the steps needed for implementing numerically the designed mathematical strategy. It is clear that all the added sources of error during the implementation will prevent the strategy to be optimal, but our goal is to test if it is still a good strategy, and how does it perform compare with simpler strategies. To be consistent with the hypothesis assumed in [3], the experiments will evaluate the logarithm of the wealth obtained in a fixed time period, when considering a portfolio composed by one risky asset and one risk-less asset of with fixed return rate. For easy reference we will call the optimal resistance strategy *ResistOpt.*

As a benchmark we will choose two simpler strategies: the optimal strategy for the classical Black and Scholes model (which we will denominate *Classical*) that may be found as $\pi = (\frac{\mu_0 - r}{\sigma^2})$ and a basic technical analysis *Tech.An.* strategy resulting from a combination of a five days moving average and a “selling when resistance is reached” order. The moving average test simply calculates for each trading day the average of the previous five trading days. If the average is greater than the last value, the order is to “sell”. Otherwise, the order is to “buy”. The “sell” and “buy” orders are represented as strategies with all the current wealth invested either in the risk-less or the risky assets respectively.

We will simulate the stochastic paths for a time interval with two subintervals.The first interval, that we will denominate *information time*, will be used to perform the resistance detection and the parameter estimation. The results of this process will be assumed constant during the second time interval, the *test time*, in which the investment is simulated and tested.

The tests we will perform in this section will be run over simulated series with fixed sets of parameters and downcross law number as shown in Table 1.6. For reference purposes, the results for a particular chosen set will be named by parameter and downcross set combination.

All the tests will evaluate the result of 1000 simulations.

1.3.1 Test with all the parameters known

We will first perform two tests assuming that we know the true values of all the parameters. The objective of this test is to fix references to understand the utility loss at each step.

First, let us assume an unbounded strategy. This means that we are allowed

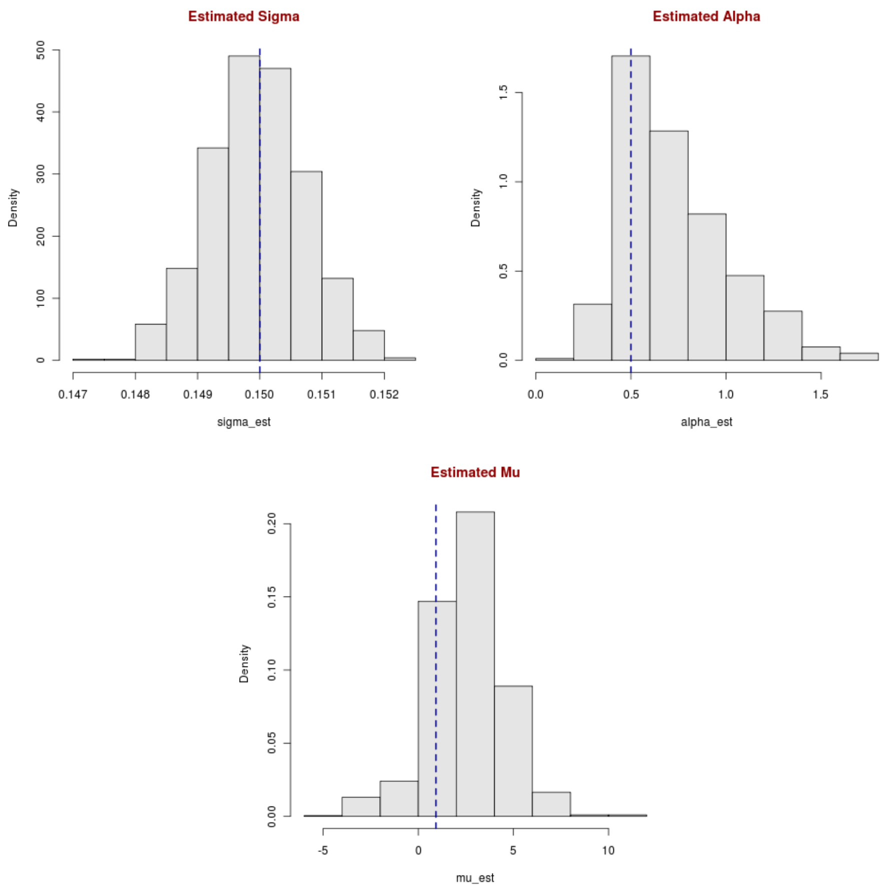

Figure 1.3: Estimation results. Up to the left: The estimation of σ gives good unbiased results. Up to the right: The estimation of α has its mode in the real value of 0.5, but the estimation is positively biased. The dispersion is important. Down: The estimation of μ has a mode and mean above the searched value. The dispersion is very important.

to borrow any amount of money either by being short on the risky asset or on the risk-less asset, in order to invest all this amount in the other asset. Our interest in this test is due to the fact that the strategy was derived under this assumption.

The results are summarized in Table 1.7. Note that the obtained wealth when using *ResistOpt* is more than twice the wealth obtained with *Classical*. In this case, the comparison with *TechAn* is unfair, since we did not give any rule for short positions in this strategy.

Clearly, the unbounded case is not representative of a real world, given that in real financial series there are restrictions regarding the maximum value an investor may actually borrow. In all the remaining comparisons, we will focus in a more realistic scenario: we will suppose the investor will not borrow at all. In terms of the allocation strategy, this traduces in restricting that the

Parameter	Value
μ_0	0.15
σ	0.15
r	0.02
α	0.5
ϵ	0.066
Information days	180
Test days	180

N	Probability
0	0.1
1	0.2
2	0.3
3	0.2
4	0.1
5	0.1

Table 1.6: Left: Set of parameters for simulated data testing. Right:Probability law assumed to rule the number of downcrossings

Classical	Tech. An.	ResistOpt
15.34 ± 4.13	2.50 ± 0.45	36.74 ± 4.54

Table 1.7: Results when all the parameters are known and an unbounded strategy is supposed.

amount invested in the risky or risk-less asset must be non negative. Since we are assuming self financed investment, this in turn implies these quantities are between $[0, 1]$.

The obtained results when the restriction over the admissible strategy set is considered is shown in Table 1.8. Clearly, a great part of the wealth generated before was due to the possibility of being short. In any case, the *ResistOpt* is clearly much profitable than the other two strategies. In the bounded case, the comparison with the *TechAn* strategy is possible, but seems to be the less profitable of the three strategies.

Classical	Tech. An.	ResistOpt
3.88 $\pm$0.67	2.49 $\pm$ 0.48	5.61 $\pm$ 0.68

Table 1.8: Results when all the parameters are known and the strategy is suppose to lie in the interval $[0, 1]$

1.3.2 Test with unknown parameters and known downcross law

Let us now consider the case in which some or all the parameters are unknown. We will split this case in some sub-cases in order to extract as much information of the simulations as possible.

- Assume first we know the right value of the resistance, as well as the downcross law. In this case, we want to observe the influence of estimating the parameters σ, α and μ. the results are shown on Table 1.9. As can be seen, the advantage of using the optimal strategy reduces in more than one unity with respect to the previous test. However, with a confidence of more than 99% the result of applying the ResistOpt strategy is greater than or equal to the classic strategy.

Classical	Tech. An.	ResistOpt
3.75 ± 0.57	2.40 ± 0.46	4.42 ± 0.55

Table 1.9: When parameters are unknown, the ResistOpt strategy still has better results than the other two, although the margin decreases importantly

- Now, suppose we need to detect first the resistance. We apply the algorithm with $\gamma = 7.79$, $S_0^+ = 1$ and two touches. As we can see from the results in Table 1.10, the final wealth obtained with the *ResistOpt* strategy reduces even further, but as before is still greater or equal than its competitors in the 99% of the simulated cases, and strictly better in the 25% of the cases.

Classical	Tech. An.	Optimal Resist.
3.6 ± 0.62	2.16 ± 0.45	4.06 ± 0.60

Table 1.10: Results with resistance detection and unknown parameters.

1.3.3 Test with unknown downcross law

Some interesting tests result from simulating not knowing the exact downcross law. For the following tests, we will suppose that the estimated downcross law is one of the following:

N	A	B
0	0.0	0.5
1	0.0	0.3
2	0.7	0.2
3	0.3	0.0
4	0.0	0.0
5	0.0	0.0

Table 1.11: Probability laws representing errors in the estimation of the downcrossings law

The coefficients of this probability laws were chosen to represent two types of errors. As can be seen when comparing Tables 1.6 and 1.11, the set A represents a downcross law that preserves the mean but with a different structure with respect to the assumed downcross for simulating, while set B have different mean and structure and thus represent a bigger miss-parametrization.

- When we suppose that all the parameters and the resistance position are known and apply different laws than the one simulated, we obtain, as expected, poorer results. However, when the law A is assumed, the results are very close to the ones obtained with the right law, while underestimating or overestimating the number of downcrosses may diminish drastically the final wealth. The simulation show results as summarized on Table 1.12.

Estimated Law	Classical	Tech. An.	Optimal Resist.
A	3.48 $\pm$0.61	2.56 $\pm$ 0.46	5.16 $\pm$ 0.61
B	3.71 $\pm$0.66	2.33 $\pm$ 0.47	3.70 $\pm$ 0.66

Table 1.12: Results with unknown law and complete parameter information

- If the parameters must be estimated and the resistance detection algorithm is used, it is necessary to sacrifice some more wealth. Table 1.13 summarizes this condition. Note that when the law with mean closer to the simulated one is used, the results are similar to those obtained with the full knowledge. The gap closes almost completely when the wrong law is used.

Estimated Law	Classical	Tech. An.	Optimal Resist.
A	3.11 $\pm$ 0.58	2.37 $\pm$ 0.45	4.22 $\pm$ 0.56
B	3.65 $\pm$0.56	2.46 $\pm$ 0.45	3.79 $\pm$ 0.55

Table 1.13: Results with unknown law and estimated parameters

1.3.4 Time consumption

A final remark with respect to the time spent during the simulations must be made. In the case in which is necessary to simulate the path, estimate the parameters, detect the resistance, apply the strategy and calculate the wealths, one entire iteration including all this steps takes around 688 ms in a common laptop with 3GB in RAM and an Intel Core 2 Duo T7250, with the estimation process being the most costly activity taking around 87.5% of the time per iteration. Therefore, the algorithm is fast enough to be used even for intra-day trading.

Chapter 2

Working with real data

2.1 Resistance detection

On section 1.1 we introduced some parameters to modulate the resistance detection algorithm. Our interest is to fix this parameters in such a way that S_0^+ models the small noise around a fixed price value, while γ should represent a significant change of price to understand that we are dealing actually with two different values.

Our intention is to define this two parameters as a function of the series itself. We lack a mathematical argument to define clearly a rule for determining this parameters since they must be calculated in a first step, even before a resistance presence is found. Based on several tests performed upon simulated and reals series, and assume we are detecting the resistance in the interval $[t_0, t_N]$ we propose to define

$$S_0^+ = Q_2(\Delta Z_t) \tag{2.1}$$
$$\gamma = \max(\Delta Z_t) \tag{2.2}$$

for for $t \in [t_0, t_N]$, where Q_2 accounts for the median function.

As shown in Table 2.1 This selection of parameters applied to the simulated data gives better results than the detection without γ (see Table 1.2, but imply a lost in right detection rate compared with the final results obtained in 1.1.

	$\eta = 2$	$\eta = 3$	$\eta = 4$
Detection Rate	99.627%	87.832 %	70.0483 %
Right detection rate	20.47%	57.359%	71.0345 %
Avg. resistance detected	95.42	99.111	100.218

Table 2.1: Results for the detection resistance algorithm with parameter given by statistics over the data themselves.

On section 2.3 we will present some results of applying the algorithm to different real series.

2.2 Downcross law

As was mentioned before, the estimation of the downcross law is a delicate issue for the success of the algorithm. The simplest way to estimate the downcross law is to determine the number of times a detected resistance is touched for historic data. In this case we are supposing that the past behavior will rule the future behavior of the downcross law.

If the number of available observations is not big enough, in order to compensate the loss of data one can assume that the downcross law is similar between observations of the same industry or group. For example, if we need more data to determine the downcross law of a company like Pfizer (belonging to pharmaceutics) it would be advisable to include data from other companies with similar profiles (for example Merck Serono S.A.).

2.3 Tests

We choose some real financial series to perform the tests in this part. To evaluate the resistance behavior for different kinds of assets, we consider one index (Dow Jones), one commodity (WTI oil prices) and some stocks coming from different industries (one energy and finances - General Electric Co., one technology - IBM, one traditional - Coca Cola Company, and one pharmaceutical - Pfizer).

The used information is freely available on the Internet. In this case we will work with the adjusted prices available at YAHOO FINANCE. The series are composed by daily close prices for almost 20 years, more specifically from January 2nd 1990 to June 30 2009, representing 4873 observation for each series.

For each of this series we will first get some statistics of resistance finding and their corresponding number of touches. The idea is that we cannot test the wealth if the resistance found does not present sufficient resistance persistence.

2.3.1 Resistance finding

In this part, we tested the resistance algorithm in the given financial series using a window with approximated size of one year. A wider window may be applied but it poses the danger of forecasting with respect to very old data. The resistance algorithm uses the parameters given by 2.1 and were obtained for $\eta = 2, 3, 4, 5$, or equivalently for 1,2,3 and 4 downcrosses.

The results are not very encouraging for the implementation of the strategy. The reason is that for the resistance algorithm to be able to detect something, one downcross must already pass. Therefore, even for the financial series with the most persistent resistance amongst the ones tested (GE stock)there is only a 12.47% of days with two downcross resistance, which traduces in an effective space for taking profit of the algorithm of just the 6% of observations. Clearly,

Downcrosses	D.J.	WTI	GE	IBM	C.Cola	Pfizer
0	56.69	52.30	47.13	65.88	53.80	47.74
1	37.92	40.07	35.25	29.23	35.53	48.45
2	4.83	6.43	12.47	4.62	8.82	3.71
3	0.45	1.14	3.93	0.28	1.79	0.11
4	0.11	0.04	1.23	0.00	0.06	0.00

Table 2.2: Observed law of downcrossings according to the used algorithm. Numbers rounded to the second decimal place

this small space is insufficient for making the strategy appealing compared with simpler ones.

Chapter 3

Conclusions and further research

During the internship we had the opportunity to understand and test an optimal strategy derived from a mathematical model similar to the Black and Scholes model but with resistance presence. We were able to implement the derived optimal strategy for resistance presence trough the use of simple and fast enough algorithms. The numerical simulation tests performed to compare the results with some simpler strategies showed that the resistance optimal algorithm outperformed the results of the optimal strategy for a classical Black an Scholes model, and those of applying a moving average with resistance (technical analysis technique), even when simple estimation procedures where used. Therefore, there is still room to increase the performance by finding estimation algorithms of higher precision for parameters as the volatility and the drift.

However a chosen set of financial series failed to show a persistent resistance detection, meaning that although resistance was frequently found, the number of "touches" associated with a resistance value was low. In practice this means that the application does not bring significant difference with respect to the classical Black and Scholes model optimal strategy, limiting the possibilities of take profit of the new strategy.

To extend the applicability of the strategy, it would be interesting to generalize the derived mathematical model to *resistance lines* in which the resistance is allowed to show a slope. This model may be applied to a wider range of financial series, as may be deduced from visual observations made during the real financial series tests. It is also interesting to complement this study with the corresponding one for *support lines* since both measures are complementary information for deciding an investment strategy.

Appendix A

C++ Library

In what follows, we present briefly the main functions and routines that support all the presented work. The basic class for which the functions are define is the `StochasticPath` class, which can be seen as an array of times and data.

A.1 Assigning values

- `int AssignVec(int n, double* t, double* d, int pos=0)`
  Assigning a vector of times (*t) and data (*d) to a StochasticPath from the position pos. n denotes the size of the vectors.

- `int AssignOne(int pos, double t, double d)`
  Assigning the element in the position pos of the StochasticPath

- `int AssignOne(double MyTime, double d)`
  Assigning the data d to the element corresponding to the time MyTime of the StochasticPath

- `int Assign (StochasticPath *B)`
  Assigning one StochasticPath to another one. Both must have the same size.

A.2 Basic mathematical operations

- `int exp(StochasticPath *A)`
  Finding the exponential of an StochasticPath: the exp function is applied to each data

- `int log(StochasticPath *A)`
  Finding the natural logarithm of an StochasticPath: the log function is

applied to each data

- `int operator+= (double k)`
  Summing a constant term to each data in the StochasticPath

- `int operator*= (double k)`
  Multiplying a constant term to each data in the StochasticPath

- `int ScaleTime(double k)`
  Scaling (Multiplying a constant scale to) the time of an StochasticPath

- `int ShiftTime(double k)`
  Shifting (Adding a constant phase to) the time of an StochasticPath

- `int sum(StochasticPath *A, StochasticPath *B, StochasticPath *Ans)`
  Summing two StochasticPaths. The function receives pointers to both stochastic paths and to the variable in which the answer will be saved. The sum is only possible if the StochasticPaths have approximately the same time vector (a difference less than 10 x FILEPRECISION)

A.3 Extracting information

- `int DataCount(double d, double epsilon, int pos_ini, int pos_fin)`
  Counting the number of positions for which the StochasticPath has values between [d-epsilon/2, d+epsilon/2] considering the interval pos_ini, pos_fin.

- `int FindValue(double dat, int CompType, int IniPos, int FinPos)`
  Finding the first occurrence in a stochastic path of a value equal to (TypeIDEqual), greater than (TypeIDGreater), greater than or equal to (TypeIDGreaterOREq), less than (TypeIDLess) or less than or equal to (TypeIDLessOREq), in the interval [IniPos, FinPos]

- `double GetData(int pos)`
  Returning the pos-th data in the StochasticPath. If pos is greater than the size it returns the last available position.

- `double GetData (double MyTime)`
  Returning the data associated with the given time. If the time is not included in the StochasticPath an error is produced and zero is returned.

- `int GetPosMax(int IniPos, int FinPos)`
  Finding the position of the maximum of a stochastic path between the positions IniPos and FinPos

- `int GetPosMin(int IniPos, int FinPos)`
  Finding the position of the minimum of a stochastic path between the positions IniPos and FinPos.

- `int GetSize ()`
  Returning the size of the vectors in the StochasticPath.

- `double GetTime(int pos)`
  Returning the pos-ieme time in the StochasticPath. If pos is greater than the size it returns the last available position.
  `bool isempty()`
  Returns true if an StochasticPath is empty

A.4 I/O functions

- `int Load(char *MyFileName, char sep=';')`
  Reading the Stochastic Path in the file with complete path MyFileName.

- `void Print()`
  Printing the content of a StochasticPath in the screen

- `int Save(char *MyFileName, bool add=false, char sep=';')`
  Writing the Stochastic Path in the file with complete path MyFileName. If add is false (default) the information is overwritten. It is added otherwise.

A.5 Path generation

- `void BM (double t0, double x0, double mu, double sigma, double tf)`
  BM generates one path of a Brownian motion between the time t0 and the time tf, with initial value t0 and n sample points (including extremes). The jumps are given by a normal distribution with mean mu·dt and variance sigma^2·(t0[i]-t0[i-1]).

- `void BouncingBM (double t0, double x0, double Alpha, double Epsilon, double mu, double tf, int NumDown)`
  BouncingBM generates one path of the stochastic process of the article, between the time t0 and the time tf, with initial value t0 and n sample points (including extremes). The higher level of the resistance is Epsilon

while the fixed level below is Alpha. The number of downcrossings to Alpha needed to allow the distribution to go to Epsilon is NumDown. The jumps are given by a normal distribution with mean rdt and variance (t0[i]-t0[i-1]).

A.6 Probability distributions

- `void rnorm (double x, double mean, double var, int n)`
  Generating n random numbers normally distributed with mean mean and variance var, by Marsaglia polar method. They are saved in the vector x
- `double ProbGeometric (int n, double par)`
  Probability function for a variable distributed as geometrically. The parameter p, the probability of the single event is pointed by par. The variable is n.
- `double CumGeometric (int n, double par)`
  Cumulative probability function for a variable distributed as geometrically. The parameter p, the probability of the single event is pointed by par. The variable is n.
- `int GenGeometric (double par, double)`
  Generate numbers distributed with a geometric law
- `double ProbUnif (int n, double initial, double final=NULL)`
  Probability function for a variable distributed discretely uniform in the interval [floor(initial),floor(final)]
- `double CumUnif (int n, double initial, double final=NULL)`
  Cumulative probability function for a variable distributed discretely uniform in the interval [floor(initial),floor(final)]
- `int GenUnif (double initial, double final=NULL)`
  Generate numbers distributed uniformly in the interval[floor(initial),floor(final)]
- `double ProbFinite1(int n, double MySize, double probs)`
  Probability function of finite support (interval [0, MySize-1]) and probability values given by the user in the pointer probs
- `double CumFinite1(int n, double MySize, double probs)`
  Cumulative probability function of finite support (interval [0, MySize-1]) and probability values given by the user in the pointer probs
- `int GenFinite1(double MySize, double probs)`
  Generating random numbers following a probability function of finite support (interval [0, MySize-1]) and probability values given by the user in the pointer probs

A.7 Technical analysis tools

- `int DownCrossIndex(double data_ref, double epsilon, double lower_ref, int pos_ini, int pos_fin)`
  Obtaining the index for the events of downcrossings in the given position interval: this index would be -1 if only upcrossings, 2n if the stochastic path is downcrossing after touching the lower_ref n times and 2n+1 if it is upcrossing after touching the lower level n times.

- `double MovingAv(int right_pos, int wsize)`
  Calculating the moving average of a window of size wsize with right extreme in right_pos. On error returns 0.

- `bool GetResistence(double& resistance, int Touches, double MyEps, double MyOut, int pos_ini, int pos_fin)`
  Calculating the value of a resistance level using the last part of the data between pos_ini and pos_fin with a tolerance of MyEps. The algorithm search for a given number of Touches that are counted if a data outside MyOut si encountered between a pair of them. The resistance variable is returned by reference. Returns TRUE if a value was found or FALSE otherwise.

- `bool GetSupport(double& support, int Touches, double MyEps, double MyOut, int pos_ini, int pos_fin)`
  Calculating the value of a support level using the last part of the data between pos_ini and pos_fin with a tolerance of MyEps.The algorithm search for a given number of Touches that are counted if a data outside MyOut if encountered between a pair of them. The support variable is returned by reference. Returns TRUE if a value was found or FALSE otherwise.

Bibliography

[1] A. Borodin and P. Salminen, *Handbook of Brownian Motion - Facts and Formulae*. Probability and Its Applications, Birkhäuser Verlag, Basel, 1996.

[2] D. Lamberton and B. Lapeyre, *Stochastic Calculus Applied to Finance*. Chapman & Hall/CRC, Boca Raton, 1996.

[3] B. Bérard Bergery, C. Propheta and E. Tanré, Mathematical Model for Resistance and Optimal Strategy. No. 524 in *FINRISK working paper series*, March 2009.

[4] M. Tinghino, *Technical Analysis Tools: Creating a Profitable Trading System* Bloomberg Press, New York, 2008.

[5] S.B. Achelis *Technical Analysis from A to Z*, 2nd Edition, McGrawHill Professional, 2000.