

**COMPARISON OF TRAINING
PERIODS FOR
STOCK MARKET
PREDICTION**

DIPLOMA PROJECT

1.NOV.1999 - 29.FEB.2000

THOMAS MEYER

IIC

ETH ZÜRICH

SUPERVISOR: PROF. DR. GASTON GONNET

INSTITUTE OF SCIENTIFIC COMPUTING

ABSTRACT

In practice, there exist three different "religions" about stock market prediction. First of all, there is the *Random Walk* hypothesis, which basically says that the stock market is not predictable at all. In the second theory - the *Fundamental Analysis* - forecasts are based on economic data. Finally, there is the *Technical Analysis* approach, which is trying to predict future price changes using only historical prices and volumes. In this work, we are studying the third approach.

We consider twenty different models based on various technical data. These models depend also on a small number of parameters. In our prediction process, we try to maximize our gain by making one decision every day - either buy or sell. To make the decision at a certain day, we use a model whose parameters are optimized to yield a maximum gain over a particular "training time" - the most recent 5, 10, 25, 50, 75 or 125 past days.

We test the models and prediction process with various stocks over a simulation time of typically half a year. One goal is to find the optimal training time for a certain model. In general, we found that we cannot expect to find a certain model with a certain training time that can be applied to any stock, as every stock is unique in its behaviour, for example different type, activity, popularity, investors, etc....

The tests also showed that it is rather difficult to outperform the *buy & hold* strategy if the stock performance is good. Statistical analysis shows that random models (models whose decisions are randomly buy or sell) have a mean gain of about half that of the *buy & hold* strategy. More importantly, the gain produced by random models typically has a high standard deviation; this makes it difficult to show that a given (presumably non random) model is not just good on the test-set by coincidence.

From the mathematical point of view, *Technical Analysis* is more like voodoo. On the one hand, it seems that the indicators of the *Technical Analysis* approach are rather bad and are not able to outperform the *buy & hold* strategy in general. On the other hand, forecasts could become true if there exist many investors who believe in the indicators - as a kind of a self-fulfilling prophecy. In any case, it is still unknown whether short-term price movements in the stock market are predictable or not.

Although the models we choose here are based on the *Technical Analysis* approach (mainly because data procurement was easier) the main focus of this work is on the prediction and maximization methods; these methods remain exactly the same also for models based on fundamental data or even other kinds of prediction problems.

The computationally most expensive part of prediction problems is the maximization of score-functions. Score-functions return a score (in our case the gain) for the simulation with certain model parameters on the training set. In our case we need to find a maximum of the score-function every time we make a single prediction: the model parameters need to be re-optimized over the training time.

Such score-functions are very special in a way that all efficient standard maximization methods fail - the functions are constant on their arguments (the gradient at almost all points is 0). To guarantee that we have found the global maximum of such a function, we must visit every single plateau.

For functions in one dimension, there is a known algorithm that has running time linear in the number of plateaus. For linear models, we have developed an efficient algorithm that is also able to find the global maximum in two dimensions. We propose an extension of this algorithm to any dimension.

However, for functions that have a huge number of plateaus, it may be impossible to visit every single plateau within a reasonable amount of time (depending on computational resources). To handle these cases we developed a heuristic algorithm based on spirals in two dimensions and on hyper-spheres in any dimension. A comparison with other maximization methods shows that this heuristic algorithm is quite good and fast.

ACKNOWLEDGEMENTS

This work is a diploma project at the Institute of Scientific Computing at the department of computer science at the Swiss Federal Institute of Technology (ETH) Zurich. It was created between 1st November 1999 and 29th February 2000.

I am especially grateful to Prof. Dr. Gonnet who gave me the opportunity to work on this project. In addition, I would like to thank to Dr. Chantal Roth-Korostensky and Arne Storjohann for their willingness to assist me in various ways.

Of course, any shortcomings and mistakes are solely my responsibility.

Thomas Meyer
(Thomas@MrMeyer.com)
94-910-007

28.Feb.2000

TABLE OF CONTENTS

1. INTRODUCTION TO STOCK MARKET PREDICTION	6
1.1 RANDOM-WALK HYPOTHESIS - BUY & HOLD	6
1.2 FUNDAMENTAL ANALYSIS - BUY LOW, SELL HIGH	6
1.3 TECHNICAL ANALYSIS - BUY HIGH, SELL HIGHER.....	7
1.3.1 Indicators.....	7
1.3.1.1 Moving averages	7
1.3.1.2 MACD Indicator	10
1.3.1.3 RSI - Indicator.....	11
1.3.1.4 Stochastic Oscillator.....	11
1.3.1.5 PRoC momentum indicator	12
1.3.1.6 On-Balance Volume indicator	13
1.3.1.7 Further theories & indicators	13
2. PREDICTION PROCESS	14
2.1 SIMPLIFIED ASSUMPTIONS	15
2.2 MODELS	15
2.3 SCORE FUNCTION	16
2.3.1 Upper limit for the score.....	17
2.3.2 Unique plateaus	18
2.3.3 Upper limit for the number of plateaus in the score function.....	18
2.4 VALIDATION.....	19
3. LINEAR MODELS	22
3.1 LINEAR DECISION MODELS	22
3.1.1 Relationship between linear decision models and perceptrons	22
3.1.2 Upper limit for the number of plateaus for linear decision - models.....	23
3.2 LINEAR ONE-PRICE MODELS	25
3.2.1 Upper limit for the number of plateaus for linear one-price models	26
3.2.2 Avoiding line segments	28
3.3 LINEAR BUYPRICE/SELLPRICE MODELS.....	29
3.3.1 upper limit for the number of plateaus for linear buy/sellprice models.....	29
3.3.2 Special case: policies with independent parameters	30
3.3.3 Avoiding line segments	31
4. MAXIMIZATION METHODS.....	32
4.1 MAXIMIZATION IN ONE DIMENSION	33
4.1.1 Complexity analysis	33
4.2 MAXIMIZATION IN 2 DIMENSIONS.....	34
4.2.1 Spiral - method	34
4.2.1.1 Choosing random start points	34
4.2.1.2 Properties of the spirals	34
4.2.2 Adaptive Squares method	35
4.2.2.1 Rules for splitting a square into 4 sub-squares:	36
4.2.2.2 Algorithm to detect crossings of lines inside a square	37
4.2.2.3 Complexity analysis	38
4.2.3 Comparison of the Adaptive Squares & Spiral method	39
4.3 MAXIMIZATION IN N-DIMENSION.....	39
4.3.1 Random directions	39
4.3.1.1 Generation of d-dimensional random directions.....	40
4.3.1.2 Box-Muller method.....	40
4.3.2 Hyper-spheres method.....	41
4.3.3 Hyper-cubes method	41
4.3.3.1 Enumeration of the corner points of the hyper-cube.....	42
4.3.3.2 Enumeration of the sub-cube corners.....	42
4.3.3.3 Construction of sub-hypercubes-ID's	43
4.3.3.4 Construction of a sub-hypercube by its ID	43
4.3.4 Simulated Annealing.....	44
4.3.5 Comparison the n-dimensional maximization methods	45

5. MODELS & RESULTS	46
5.1 MODEL 1: MEAN & STANDARD DEVIATION OF PAST PRICES	46
5.2 MODEL 2: COMPUTING THE TREND BY A LINEAR LEAST SQUARES FIT	48
5.3 MODEL 3: ESTIMATION OF HIGH ₁ AND LOW ₁ BY EXTRAPOLATION	49
5.4 MODEL 4: OPTIMAL COMBINATION OF DIFFERENT TRENDS	50
5.5 MODEL 5: P% FILTER RULE	52
5.7 MODEL 7: COMBINATION OF INDICATORS FROM <i>TECHNICAL ANALYSIS</i>	55
5.8 MODEL 8: INDICATORS FROM <i>TECHNICAL ANALYSIS</i> (SECOND MODEL)	57
5.9 MODEL 9: RELATIONSHIP VOLUME / TREND	58
5.10 MODEL 10: DECISION - MODEL.....	60
5.11 MODEL 11: BEST TRADING DAYS.....	61
5.12 MODEL 12: ONE-PRICE MODEL.....	62
5.13 MODEL 13: PREDICTION OF SUPPLY AND DEMAND	63
5.14 MODEL 14: A RANDOM BASED MODEL	65
5.16 MODEL 16: PATTERN MATCHING.....	66
5.17 MODEL 17: MAJORITY DECISION OF INDEPENDENT MODELS	68
5.18 MODEL 18: SUPPLY AND DEMAND AS FUNCTIONS OF COSINE AND SINE.....	70
5.19 MODEL 19: EXTENSION OF MODEL 1	72
5.20 MODEL 20: STATISTICAL APPROACH.....	73
6. IMPLEMENTATION.....	75
6.1 PROGRAM DESIGN	75
6.2 EXTENSION WITH NEW MODELS.....	76
6.3 EXTENSION WITH NEW STOCKS.....	77
6.4 VERIFICATION OF THE PROGRAM.....	77
7. CONCLUSIONS AND FUTURE WORK.....	79
8. REFERENCES	81
9. APPENDIX	82

1. INTRODUCTION TO STOCK MARKET PREDICTION

1.1 Random-Walk hypothesis - buy & hold

One opinion about stock market prediction is the *Random-Walk* hypothesis. The strong version of the *Random Walk* hypothesis states that all price movement is the result of supply and demand in a varying degree of knowledge (some people are well informed, some people are badly informed). As the force of supply and demand is unpredictable, price changes are also unpredictable. From that reason, it does not make sense to try to predict price changes in the stock market.

A weaker version of this hypothesis admits the existence of trends, but it still denies the possibility to outperform the *buy & hold* strategy in general with any kind of strategy.

If the strong version of the *Random Walk* hypothesis was true, one has to accept the idea that it does not matter where to invest the money. If it is all random, no degree of knowledge can possibly improve the gain, and all the experts would be paid for nothing.

The best strategy according to the *Random Walk* hypothesis is to lower the volatility (and therefore the risk) by diversifying the portfolio and to apply the *buy & hold* strategy.

One indication against the strong version of this theory is the fact that companies whose stocks rise over many decades tend to be well managed, are earning consistent profits and paying dividends; and there is nothing random about this.

1.2 Fundamental Analysis - buy low, sell high

«The major direction of the market is dominated by monetary considerations, primarily Federal Reserve policy and the movement of interest rates.»

Martin Zweig

The general idea behind the *Fundamental Analysis* is to predict future values of a certain stock by the fundamentals of the company. Any change in the overall economic situation can have an impact on industrial sectors and companies. Therefore, it is necessary not only to study the internals of a company, but also the overall economic situation and the industrial sectors. Table 1.1 lists some important factors:

economic situation	Different impact on industrial sectors	Company
<ul style="list-style-type: none"> • economic growth • interest rates • inflation • exchange rates • consumer price index • unemployment rate • other stock markets • taxes • ... 	<ul style="list-style-type: none"> • oil/energy prices • raw material prices • new technologies • ... 	<ul style="list-style-type: none"> • dividends • P/E ratio (nowadays maybe rather the sales increase) • cash-flow • management quality • competitive position of the company within the industrial sector • AAA • ...

Table 1.1 economic factors that can influence stock prices

Every change in the overall economic situation can have a different impact on the different industrial sectors. For example, if inflation is rising, interest rates are also rising. While most of the stocks will be under pressure, gold stocks become popular and tend to rise. But also within the same industry, there can be a different impact from changing environment variables. For example if oil prices rise, many industrial sectors will suffer from it, also the car industry. However, small cars might become more popular and therefore it could even be a good sign for producers of small cars.

One of the difficulties of the fundamental approach is to collect all the important data and combine the data into a good model. The correctness of a company's internal data will also be a problem as companies are trying to manipulate their internal data within the legal range to be as attractive as possible to investors.

A further disadvantage of the fundamental approach is that there is sometimes a huge difference between the price for a certain stock and the "real" (fundamental) value of that stock. A good example for this is *amazon.com*, *Linux* or some other internet stocks.

As nowadays more and more people become investors, popularity of stocks (e.g. rumours, good news, bad news), which actually have nothing to do with the company's fundamental values, can not be neglected. And the popularity of a certain stock among investors can not easily be expressed by a number.

1.3 Technical Analysis - buy high, sell higher

«When you know absolutely nothing about the topic, make your forecast by asking a carefully selected probability sample of 300 others who don't know the answer either»

Edgar R. Fiedler about *Technical Analysis*.

In *Technical Analysis*, the main idea is that changes in stock prices can be predicted based on recent trends in stock price changes, the relationship between price and earnings, volume activity in particular stock or industry and some other indicators. A basic assumption of this approach is that all the economic information is already contained in the current stock price. Therefore people believe that the predictions can be done by only considering charts and past prices.

While the fundamental approach is making conclusions from economic data to stock prices, the *Technical Analysis* approach is taking conclusions the other way round.

There is no time horizon given to apply the methods of *Technical Analysis*: it can be applied to hours, days, months or even years (under the assumption that the indicators were good).

There exist quite a lot of indicators that produce "BUY" and "SELL" signals, but which are - from the mathematical point of view - very questionable. One of many critic points is that in contrary to weather forecasting, the *Technical Analysis* approach can cause a "self fulfilling prophecy": If there exist a lot of investors who believe in these signals, the signals become true and good. Another critic point is that technical analysts often try to prove their indicators by giving a few examples from the past where these indicators produced correct signals.

Below, we will have look at some indicators a bit more in detail:

1.3.1 Indicators

1.3.1.1 Moving averages

Moving averages are one of the oldest and most popular *Technical Analysis* tools. A moving average is an indicator that shows the average price value over a specified period of time. The most popular method to interpret a moving average is to compare the relationship between a moving average of a price with the price itself. A BUY signal is generated when the price rises above its moving average and a SELL signal is generated when the price falls below its moving average.

There exist three different kinds of moving averages:

- arithmetic moving average (all days within the period have the same weight)
- triangular moving average (linear decrease of the weights)
- exponential moving average (exponential decrease of the weights)

The number of days considered in the moving average is also variable. Normally either 200, 50 or 20 days are considered. The more days, the "slower" the moving average reacts to changes in the price.

An example (adobe):

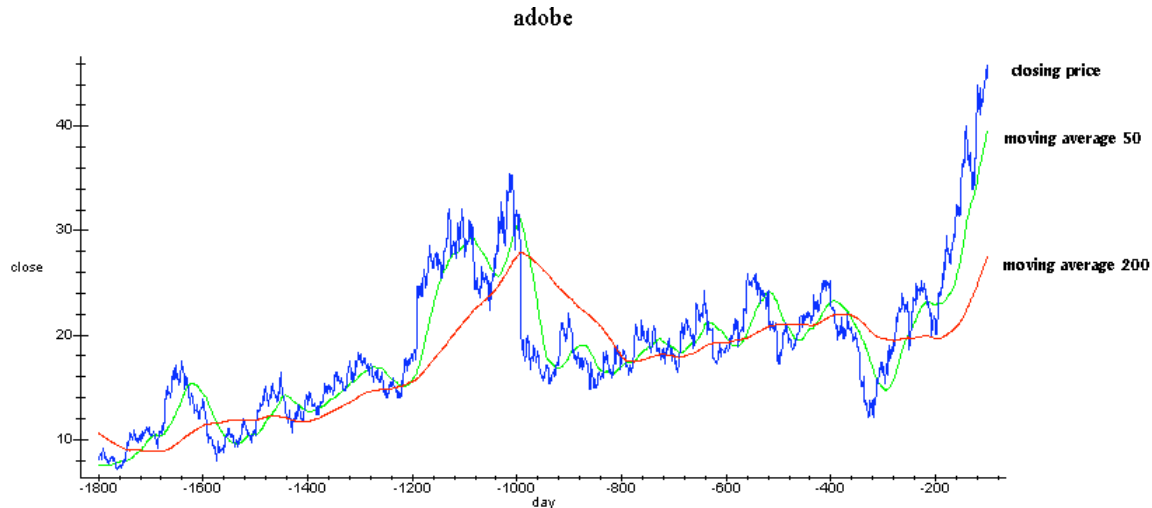


Figure 1.1: arithmetic moving averages

The more days considered in the moving average, the less signals will be generated (crossings of the price line with the moving averages line). Table 1.1 shows the result of the simulation with this strategy over the most recent 4 years:

- BUY, if the price was crossing its moving average from below at the previous day.
- SELL, if the price was crossing its moving average from above at the previous day.

STOCK	start date of simulation	end date of simulation	Buy & Hold strategy	Random Buy/Sell	arithmetic			exponential
					200 days	50 days	20 days	50 days
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 154.8 %	+ 525.8 %	+ 249.2 %	+ 319.8 %	+ 244.7 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 65.7 %	+ 196.3 %	+ 123.8 %	+ 144.5 %	+ 140.4 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 6.5 %	+ 1.1 %	+ 203.9 %	+ 132.5 %	+ 116.0 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	- 11.8 %	- 13.0 %	- 35.8 %	- 53.4 %	- 56.0 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 105.8 %	+ 21.6 %	- 8.3 %	+ 0.1 %	- 16.6 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	+ 35.1 %	+ 45.5 %	+ 24.8 %	+ 10.0 %	- 5.1 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 101.6 %	+ 37.3 %	+ 21.1 %	+ 45.1 %	+ 51.5 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 42.5 %	+ 4.5 %	+ 0.2 %	+ 9.2 %	- 9.3 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 150.1 %	+ 56.1 %	+ 23.7 %	+ 35.5 %	+ 40.6 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 152.9 %	+ 1.01 %	+ 32.0 %	- 0.1 %	+ 14.7 %

Table 1.2: result of a with the moving average strategy

Looking at table 1.2, we can see that the moving average strategy is a rather bad strategy. A somewhat amazing result can be found if we try out exactly the opposite strategy: Buying when the indicator indicates selling, and selling when the indicator indicates buying:

STOCK	start date of simulation	end date of simulation	Buy & Hold strategy	arithmetic			exponential
				200 days	50 days	20 days	50 days
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 33.1 %	+ 138.6 %	+ 98.5 %	+ 141.8 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 7.0 %	+ 41.6 %	+ 29.6 %	+ 31.8 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 97.8 %	-34.2 %	- 14.0 %	- 7.4 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	+ 11.7 %	+ 51.3 %	+ 108.3 %	+ 120.5 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 60.6 %	+112.9 %	+ 95.2 %	+ 134.0 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	- 15.7 %	- 1.8 %	+ 11.5 %	+ 29.2 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 18.1 %	+ 33.9 %	+ 11.8 %	+ 7.1 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 38.3 %	+ 44.3 %	+ 32.3 %	+ 59.6 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 83.4 %	+ 131.4 %	+ 111.3 %	+ 103.6 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 85.3 %	+ 41.8 %	+87.3 %	+ 63.2 %

Table 1.3: result of a simulation with the opposite moving average strategy

This short simulation shows that the usage of moving averages is not a good idea at all. It rarely outperforms the *buy & hold* strategy.

However, an argument against the above simulation could be that we were not taking action immediately after the crossing of the moving average line and the price line, but always only on the following day. Let us consider a model that is taking action immediately after the crossing. Assuming that we use arithmetic moving averages, the crossing of the two lines at the present day can be computed as follows:

$$\text{movAvg}_{\text{day}}^* == \text{actionPrice}_{\text{day}}$$

$$\text{movAvg}_{\text{day}}^* = \text{movAvg}_{(\text{day} - 1)} + \frac{\text{actionPrice}_{\text{day}} - \text{Close}_{(\text{day} - \text{movAvgPeriod})}}{\text{movAvgPeriod}}$$

solving for $\text{actionPrice}_{\text{day}}$ gives the following price:

$$\text{actionPrice}_{\text{day}} = \frac{\text{movAvgPeriod} \cdot \text{movAvg}_{(\text{day} - 1)} - \text{Close}_{(\text{day} - \text{movAvgPeriod})}}{\text{movAvgPeriod} - 1}$$

But the table 1.4 shows that this strategy is rather bad as well:

STOCK	Start date of simulation	end date of simulation	Buy & Hold strategy	arithmetic		
				200 days	50 days	20 days
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 317.3 %	+ 361.3 %	+ 328.1 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 321.5 %	+ 102.6 %	+ 135.2 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 85.3 %	- 26.7 %	+ 154.6 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	+ 36.0 %	- 18.3 %	+ 16.1 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 41.7 %	+ 16.7 %	+ 62.4 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	+ 102.0 %	+ 13.9 %	- 4.7 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 14.7 %	+ 34.9 %	+ 39.5 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 8.0 %	+ 106.2 %	+ 27.2 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 121.1 %	+ 3.4 %	+ 43.3 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 39.1 %	+ 2.3 %	+ 8.5 %

Table 1.4: simulation of the moving average strategy where we buy immediately after the crossing of the indicator lines

1.3.1.2 MACD Indicator

The **M**oving **A**verage **C**onvergence **D**ivergence indicator is also a popular and widely used indicator. It is a trend - following momentum indicator that compares two moving averages (a "slow" and a "fast" one) of price history. Normally, the "fast" line is the difference between a 26-day and a 12-day exponential moving average. The "slow" line is a 9-day exponential moving average of the fast line.

When the "fast" line falls below the "slow" line, a SELL signal is generated. When it rises above the "slow" line, a BUY signal is generated. Some people also consider the MACD line crossing above or below zero as BUY or SELL opportunities.

The MACD indicator is also used as an overbought/oversold indicator. When the "fast" line pulls away from the longer moving average due to a strong upward price trend, indicated by a rising MACD, price is likely overextending and will likely fall back to more realistic levels.

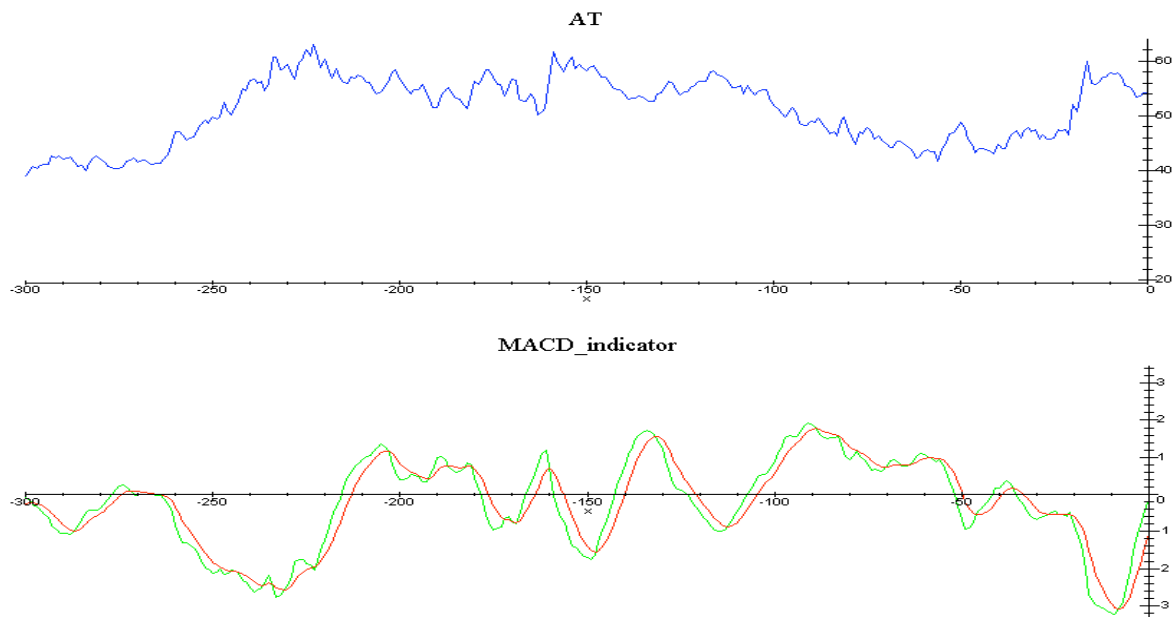


Figure 1.2: closing price of AT&T and the corresponding MACD indicator

A simulation with the MACD indicator strategy shows that this indicator is not a real good indicator either. Even the opposite strategy often seems to be better.

STOCK	Start day of simulation	End day of simulation	BUY & HOLD	Random BUY/SELL	MACD	opposite MACD
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 154.8 %	+ 179.6 %	+ 198.0 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 65.7 %	+ 28.5 %	+ 146.7 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 6.5 %	+ 19.5 %	+ 65.4 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	- 11.8 %	+ 46.5 %	- 37.7 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 105.8 %	+ 70.5 %	+ 14.5 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	+ 35.1 %	+ 0.3 %	+ 24.6 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 101.6 %	+ 11.1 %	+ 46.0 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 42.5 %	+ 50.3 %	- 3.6 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 150.1 %	+ 112.9 %	+ 34.5 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 152.9 %	+ 37.3 %	+ 38.3 %

Table 1.5: simulation with the MACD strategy

1.3.1.3 RSI - Indicator

The **Relative Strength Indicator** was originally developed by James Welles Wilder in 1978. This oscillator ranges between 0 and 100. Values over 70 (sometimes 80) are considered as overbought market, values below 30 (sometimes 20) as oversold market. BUY signals are generated as soon as the RSI-value is below 20 and rises above 20 again, and SELL signals are generated as soon as the RSI value is above 80 and drops below 80 again.

Formula:

$$RSI = 100 - \frac{100}{1 + RS}$$

$$RS = \frac{\text{average upward price change (14 days)}}{\text{average downward price change (14 days)}}$$

The average of the closing prices can be computed over the past N days, where N could be any number. Normally the RSI value is computed with $N=9$ days or $N=14$ days.

STOCK	Start day of simulation	End day of simulation	BUY & HOLD	Random BUY/SELL	RSI	opposite RSI
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 154.8 %	+ 31.8 %	+ 558.6 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 65.7 %	+ 394.0 %	- 39.6 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 6.5 %	+ 92.7 %	+ 70.5 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	- 11.8 %	- 10.6 %	+ 5.3 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 105.8 %	+ 46.4 %	+ 29.3 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	+ 35.1 %	- 14.4 %	+ 55.8 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 101.6 %	+ 40.6 %	+ 16.5 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 42.5 %	+ 4.4 %	+ 38.1 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 150.1 %	+ 132.6 %	+ 23.9 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 152.9 %	+ 65.4 %	- 2.6 %

Table 1.6: simulation with the RSI strategy

A completely random strategy seems to be better than using the RSI indicator.

1.3.1.4 Stochastic Oscillator

The stochastic oscillator is a momentum indicator developed by George Lane. Stochastics measure the relationship between closing prices and the previous maximum high and minimum low intra day prices over a specified number of days.

The stochastic oscillator consists of a fast (%K) and a slow (%D) line. A BUY signal occurs when the %K line rises through the %D line when it is below 20. A SELL signal occurs when the %K line falls through the %D line above 80.

%D - Line: moving average (3 days) of the %K Line

$$\%K - \text{Line} : 100 \cdot \frac{\text{close}_i - \text{lowestPrice}_{14}}{\text{highestPrice}_{14} - \text{lowestPrice}_{14}}$$

STOCK	Start day of simulation	End day of simulation	BUY & HOLD	Random BUY/SELL	original strategy: (Sell signals only above 80 ; BUY signals only below 20)		all crossings produce signals	
					stochastic	opposite stochastic	stochastic*	opposite stochastic*
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 154.8 %	+ 84.4 %	+ 356.9 %	+ 209.6 %	+ 169.1 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 65.7 %	+ 26.8 %	+ 152.8 %	+ 97.4 %	+ 62.4 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 6.5 %	+ 26.5 %	+ 153.9 %	- 4.3 %	+ 109.0 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	- 11.8 %	+ 92.9 %	- 49.3 %	- 31.5 %	+ 42.75 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 105.8 %	+ 67.8 %	+ 23.1 %	- 15.1 %	+ 132.0 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	+ 35.1 %	- 33.5 %	+ 88.6 %	+ 21.4 %	+ 1.0 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 101.6 %	+ 13.7 %	+ 44.1 %	+ 29.8 %	+ 24.9 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 42.5 %	+ 45.7 %	- 3.4 %	- 10.8 %	+ 62.1 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 150.1 %	+ 24.8 %	+ 126.5 %	- 27.1 %	+ 287.1 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 152.9 %	+ 57.8 %	+ 21.9 %	+ 30.4 %	+ 45.5 %

Table 1.7 simulation with the Stochastic-Oscillator strategy

1.3.1.5 PRoC momentum indicator

The Price Rate of Change momentum indicator is the ratio between the current price and the price N days ago. The most popular time periods are 12-day and 26-day price comparisons for short and mid-term indications of price momentum.

As price increases, the PRoC increases. Likewise, as the price decreases, the PRoC decreases. As such, the PRoC is said to be an excellent indicator for overbought and oversold conditions.

The PRoC indicator is an oscillator around 100. SELL signals are generated if the oscillator is crossing the $y=100$ line from above and BUY signals if the oscillator is crossing the $y=100$ line from below.

$$\text{momentum}_i = \frac{\text{close}_i}{\text{close}_{(i-12 \text{ days})}} \cdot 100$$

STOCK	Start day of simulation	End day of simulation	BUY & HOLD	Random BUY/SELL	PRoC	opposite PRoC
Microsoft	22.12.1995	10.12.1999	+ 733.3 %	+ 154.8 %	+ 210.0 %	+ 168.9 %
Apple	26.12.1995	10.12.1999	+ 216.9 %	+ 65.7 %	+ 17.3 %	+ 183.2 %
Adobe	26.12.1995	10.12.1999	+ 99.9 %	+ 6.5 %	+ 174.0 %	- 27.9 %
Boeing	03.01.1996	17.12.1999	- 2.9 %	- 11.8 %	- 33.1 %	+ 50.7 %
General Motors	03.01.1996	17.12.1999	+ 95.2 %	+ 105.8 %	+ 38.0 %	+ 47.4 %
AT & T	05.01.1996	21.12.1999	+ 22.6 %	+ 35.1 %	+ 52.9 %	- 19.8 %
Coca - Cola	05.01.1996	21.12.1999	+ 62.2 %	+ 101.6 %	+ 117.6 %	- 25.8 %
Walt Disney	05.01.1996	21.12.1999	+ 44.6 %	+ 42.5 %	+ 45.2 %	- 0.4 %
Hewlett Packard	05.01.1996	21.12.1999	+ 186.3 %	+ 150.1 %	+ 99.7 %	+ 43.4 %
McDonalds	05.01.1996	21.12.1999	+ 87.2 %	+ 152.9 %	+ 32.2 %	+ 43.4 %

Table 1.8: simulation with the PRoC strategy

1.3.1.6 On-Balance Volume indicator

The *On-Balance Volume* indicator was developed by Joseph Granville in 1963. The idea behind this indicator is that the volume should follow the trend. An increasing *On-Balance* value is considered as a good sign (BUY) while a decreasing *On-Balance* value is considered as a bad sign (SELL).

The *On-Balance* value is computed as follows:

```
if ( closei ≥ openi )
    OnBalanceValuei = OnBalanceValuei-1 + Volumei
else
    OnBalanceValuei = OnBalanceValuei-1 - Volumei
fi
```

1.3.1.7 Further theories & indicators

Above indicators were just a selection of the most popular indicators. In the *Technical Analysis* approach, there exist a huge number of different theories and indicators, but no theory has been proved yet and no indicator has been shown to outperform the *buy & hold* strategy in general. Here are a few more indicators:

<i>theories</i>	Dow theory, Fibonacci-series, Gann-theorie, ...
<i>patterns</i>	head-shoulder patterns, support / resistance lines, triangles, ...
<i>McClellan Oscillator</i>	A market breath indicator that measures the smoothed difference between advancing and declining issues on the New York Stock Exchange. It is calculated by taking the difference between a 19-day and 39-days exponential moving averages of advancing minus declining issues.
<i>new highs vs. new lows</i>	The difference between the number of issues hitting a new 52-week highs and new 52-week lows. It is said to be an excellent divergence indicator. It helps identifying changes in the market price trend.
<i>advance/ decline line</i>	cumulative difference of the number of climbing stocks and falling stocks.
<i>most active stocks</i>	ratio between climbing and falling stocks, only considering the 20 most active stocks (with highest trading - volume).
<i>put/ call ratio</i>	ratio between puts and calls. If there are more puts than calls it is a good sign, otherwise a bad sign.
<i>upside/ downside ratio</i>	ratio of volumes of stocks whose prices climbed and stocks whose prices fall.
<i>insider trading</i>	Whenever insiders of a company are buying or selling stocks of their own company, they have to report their intention and also the reason why they want to buy or to sell. All the insider trades are open to the public (e.g. http://biz.yahoo.com/t/). Although this restriction can easily be evaded in an illegal way, it is still said to be a good indicator.
<i>bid & ask</i>	Banks have the possibility to see the ratio between bids and asks for every stock at any time. With this information, it is probably rather easy to predict whether stock prices will climb in the next few hours or not.

2. PREDICTION PROCESS

The prediction process can be described as follows:

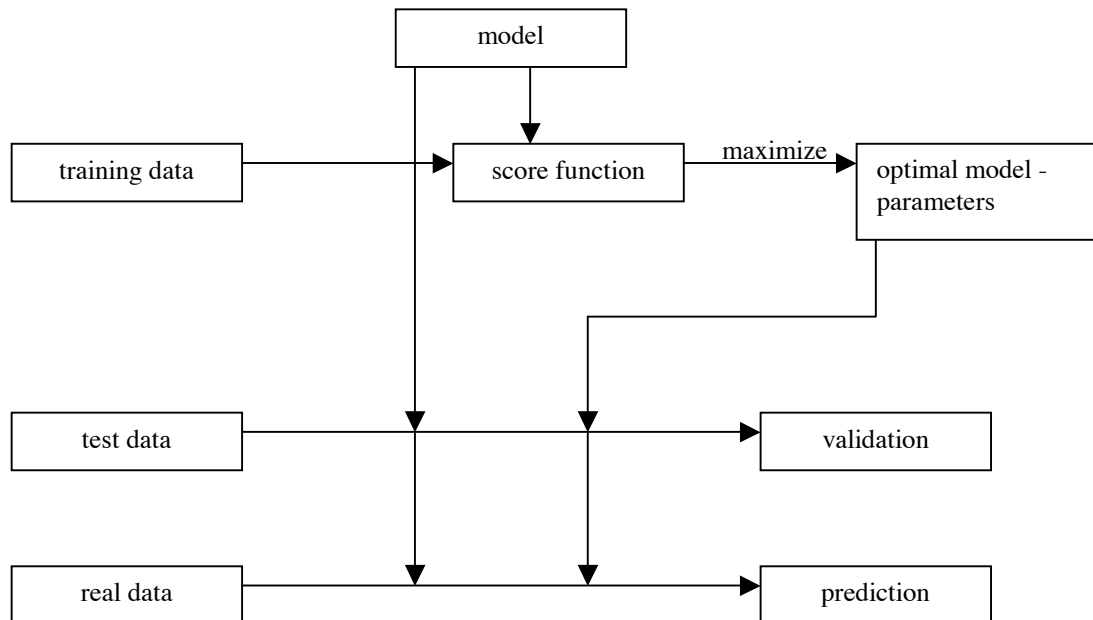


Figure 2.1: scheme of the prediction process

A model makes a decision at a certain day according to a list of parameters and input data of that day. The "model" could be a simple formula as well as a complex program. In the training period we simulate the stock market. For every day of the training, we let the model make a decision and act according to that decision. The score function computes the total gain of the simulation over the training period with certain model parameters. The higher the gain, the better the choice of the parameters. By evaluating the score function many times with different arguments, the maximization process is computing the optimal model parameters for this training period.

To validate a given model, we simulate the stock market for typically half a year:

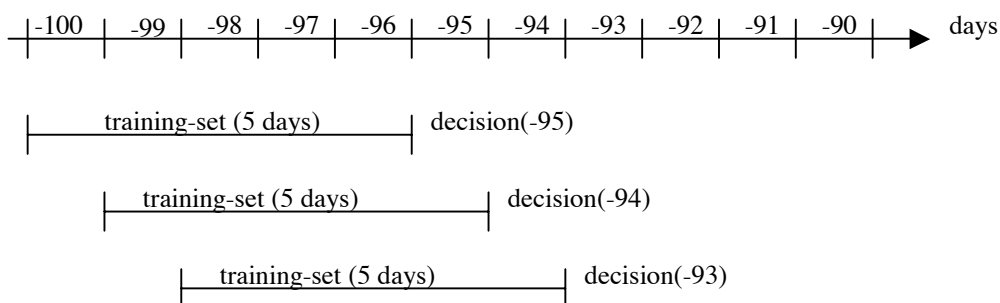


Figure 2.2: simulation scheme

In our approach, we are using a dynamic window that is moving from day to day. To make a decision at certain day, we use a model whose parameters are optimized to yield a maximum gain over a particular training time - the most 5, 10, 25, 50, 75 or 125 past days. As the training period is changing every day by the moving window, we must re-optimize the model parameters every day from new, which is computationally very expensive.

If the model turns out to be good in this simulation and it also succeeds in further validation tests, it can be applied to the real world.

2.1 Simplified assumptions

We make the following simplifications:

At most one transaction per day

The problem would remain exactly the same if we made several transactions per day or only one transaction per week or month. It might be the case that the shorter the period we are choosing, the less predictable the problem becomes. However so far, it has not been proved or disproved that our problem is predictable at all.

Only two states: {MONEY,STOCKS}

We are always buying stocks with all our money and we are always selling all the stocks we own. Mathematically seen, it can not be optimal to own both stocks and money at the same time. Every day, there exists exactly one good decision (except for the special case of no price movement). This decision is either BUY or SELL. If the best decision is BUY and we are not buying with all our money, we can not reach the maximum gain. For simplicity, we allow fractional numbers of shares.

Buying/ selling at our desired price

We assume that the stock market is liquid enough that we can always buy and sell at our desired price (unless the price is not within the price range of the day).

No Transaction costs

We assume that we have no transaction costs, as transaction costs are not part of the prediction problem.

Open-price of the current day can be used for the training

We assume that we know the open-price of day k already in advance and therefore we can use it to make our forecast for day k

2.2 Models

In general, models are formulas or programs that combine unknown model parameters with data. The goal of models is to predict processes in the real world. In our case, we use models to decide whether we should buy or sell at a certain day.

We consider models as black-boxes: The input of a model are certain model parameters, data of stock price history, the current day, and in some special cases a model-state. With this input, the model is computing an output, which depends on the type of the model: either a decision (BUY, SELL) for **decision models**, a single action-price for **one-price models** or both buy-price and sell-price for **buy/sellprice models**.

Each of the three types is described below:

decision - models

Decision models only return a decision, which can be either BUY or SELL and in some special cases DO_NOTHING. If at *day k*, the model says BUY, we are going to buy at the opening price of *day k*, if the models says SELL, we are going to sell at the opening price of *day k*.

Note for every day, there exists a good decision. If the stock price is rising that day, the decision should be BUY, accordingly if the stock price is decreasing the decision should be SELL. DO_NOTHING can only be a good decision if the stock price is neither climbing nor falling that day.

A drawback of decision-models is that we are not able to reach the maximum possible gain in any case as we are always buying and selling for the opening price.

buy-price / sell-price models

In this kind of model, we have two different policies to compute a BUY-price and a SELL-price. Depending on our current state (MONEY or STOCKS), we use either the BUY-price or the SELL-price policy. If our state is MONEY and the stock price is falling as deep as the suggested buy-price at that day, we are going to buy, otherwise we are going to keep our money. Accordingly, if our state is STOCKS, we are consulting the model for the sell-price of the current day. If the stock price at that day reaches this price, we are going to sell at that price. Furthermore, we can assume that we are never going to buy at a price higher than the opening price and we are never going to sell at a price lower than the opening price.

The buy-price returned at *day k* should be lower than the opening price of *day k+1*. And the SELL-price returned at *day k* should be higher than the opening price of *day k+1*.

Note, although the opening price of *day (k+1)* is higher than the opening price of *day k* (decision models should return BUY), we are still making a good decision if we sell our stocks at a higher price than the opening price of *day k+1*. (Accordingly for BUY)

one-price models

One-price models also return a price for every day. Contrary to the buyprice/sellprice models, there is only one policy. Depending on the current state, we consider the suggested price either as buy-price or as sell-price. As we can use the opening price of the current day for our prediction, we can make the following improvement:

$$\begin{aligned} \text{buy-price} &= \min(\text{modelprice}, \text{openPrice}) \\ \text{sell-price} &= \max(\text{modelprice}, \text{openPrice}) \end{aligned}$$

With this improvement, we have a guarantee that we never buy at a higher price than necessary and never sell at a lower price than necessary.

Although this improvement seems to be the better choice than always buying at the model-price, this improvement is a restriction to the model: In *Technical Analysis*, people sometimes believe that crossings of 2 different indicator lines (e.g. support, resistance lines) lead to a signal. Let us assume that we only want to buy if the price of the stock is crossing a certain resistance line and we are not going to buy if the price does not cross the resistance line. The resistance line itself is higher than the opening price. In that case we just want to buy at a price that is higher than the opening price.

2.3 Score function

The score-function f is a function that takes a list of N model-parameters. It returns a score (in our case the gain) for the simulation with these model-parameters on the training-set.

$$f: \mathbb{R}^N \rightarrow \mathbb{R}, \quad (p_0, p_1, p_2, \dots, p_{N-1}) \rightarrow f(p_0, p_1, p_2, \dots, p_{N-1})$$

The parameters $p_0, p_1, p_2, \dots, p_{N-1}$ are continuous. The score-function f itself is not continuous as already only a small change in one single parameter can change the decision at any day and therefore completely the score. Whenever this happens, we have a discontinuity in the score-function f . On the other hand, a change in one or even more parameters does not necessarily affect any decision and will therefore lead to exactly the same score. From that follows that the score-function f is piecewise constant on its arguments.

Figure 2.3 shows a typical score-function:

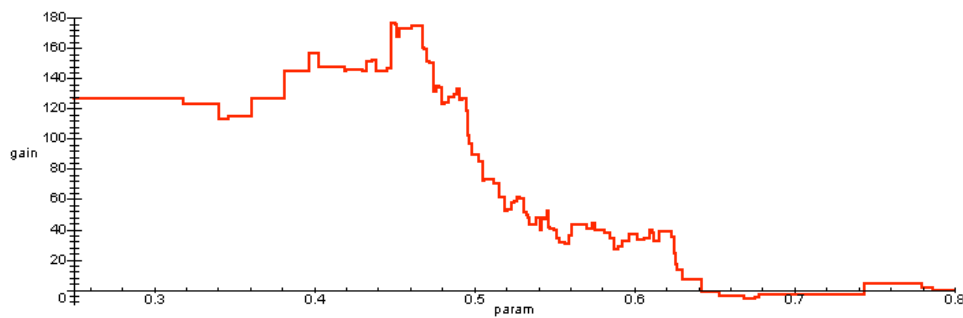


Figure 2.3: score function in one dimension

From now on, let us call a parameter range with constant score "**plateau**". The goal of the maximization process is to maximize the score, which is equal to finding the plateau with the highest score. All the parameters on that plateau lead to exactly the same decision sequence and therefore they are all optimal model parameters.

In our approach we are maximizing the gain on the training period. Basically, we could also use a different score-function, such as a function that returns the number of good decisions.

Depending on the score-function we choose, we will normally get different optimal model parameters. Choosing the number of good decisions as our score-function, the model will learn to make good decisions. Choosing the total gain as score-function, the model will try to maximize the gain, which is actually our goal.

2.3.1 Upper limit for the score

The upper limit of the gain on the training period can be computed by dynamic programming. For every day, we can have two states: MONEY or STOCKS. The idea of dynamic programming is to compute the optimal states for both MONEY and STOCKS at *day k+1*, assuming that we have optimal states at *day k*.

Whenever we buy, we are buying at the lowest price and whenever we sell, we are selling at the highest price of the day. Therefore the optimal state for STOCKS at *day k+1* is the maximum over the two possibilities of keeping the stocks at *day k* or buying stocks at the deepest price at *day k*.

The optimal state for MONEY at *day k+1* is the maximum on either keeping the money at *day k* or selling all the stocks for the highest price at *day k*.

At the very last day, we have to values for the optimal states MONEY and STOCKS. To find out the best state and the maximum gain, we can compare these two states by selling the stocks at the highest price of the very last day. The maximum we get from this comparison is the maximum possible gain on this (training) period.

It is also possible to find out the optimal sequence by backtracking, but in our case we are only interested in the maximum. Example:

	InitValues	Day[0]	Day[1]	Day[2]	...	Day[K-1]
Money [\$]	100	100	104	135	...	
#Stocks	0	4	5	5	...	
<i>High of the day [\$]</i>		30	26	27		
<i>Low of the day [\$]</i>		25	20	23		

for day $i = 0$:
 $Money[0] = 100$; (START CAPITAL);
 $Stocks[0] = 100 / Low[0]$;

for day $i > 0$:
 $Money[i] = \max(Money[i-1], Stocks[i-1] \cdot High[i])$
 $Stocks[i] = \max(Stocks[i-1], Money[i-1] / Low[i])$

At day $K-1$, we get the upper limit of the possible gain (subtracting the start capital):

$$\text{max_gain} = \text{max}(\text{Money}[K-1], \text{Stocks}[K-1] \cdot \text{High}[K-1]) - 100$$

Remark: The upper limit for decision models is significantly lower than the upper limit for models in general, as with decision models, we are always buying and selling at the opening price.

2.3.2 Unique plateaus

For the correctness of our maximization algorithms, we need to have a guarantee that all the plateaus are unique in their score; any two distinct decision sequences need to have different scores.

A correct way to solve this problem would be to compare the whole transaction sequence of the parameter points whenever two parameter points have the same score. Using this approach, we do not necessarily need unique plateaus.

In a second approach, we could have a different prime-number for every day and decision. In the simulation we are multiplying the primes according to our decisions. Multiplying these primes, we will always end up with a different number for every possible decision sequence. Once we find the same score for two different parameter points, we need to compare this number, which is the result of the multiplication of prime numbers. If two parameter points have the same number, the decision sequence must have been exactly the same and therefore the two parameter points must be on the same plateau.

Our approach is to add a very small noise (depending on the day) to the rounded (1/16) buy-price or sell-price, whenever we buy or sell. Although we do not have a guarantee of unique plateaus for sure, parameter points with the same score are lying on the same plateau with a very high probability.

2.3.3 Upper limit for the number of plateaus in the score function

As every different decision sequence in the training period leads to a different score (guaranteed by unique plateaus), we need to count the number of different decision sequences that are possible. The number of plateaus is depending on the type of model.

In the case of decision-models, we can have 2 different decisions every day: BUY or SELL. Therefore we can have at most 2^N different decision sequences, where N is the number of training days. The upper limit for the number of plateaus is therefore increasing exponentially in the number of training days.

In the case of price-models, we can have $\lceil 16 \cdot (\text{high}(\text{day}) - \text{low}(\text{day})) \rceil$ different prices (prices are rounded to 1/16). For simplicity, let us assume that we can have 40 different prices every day. The upper limit for the number of plateaus is 40^N , with N training-days. Already with just a few number of training days, this will be a really huge number.

2.4 Validation

To validate a model, we test it with various stocks (blue chips) over a simulation time of 125 days, which is about half a year. It is important that the model has never seen these test-data before. If the model is good, it is supposed to get a good result in this simulation.

We run this simulation several times, always with a different number of training days - either 5, 10, 25, 50, 75 or 125 days. In general, we cannot expect to find a model with a certain training time that can be applied to any stock, because stocks are all unique in their behaviour - investors, activity, popularity, etc....

We also compare the results with random models (models whose decisions are randomly buy or sell):

STOCK	125 Simulation days			random decisions (1000 independent simulations)	
	start date	end date	Buy & Hold	Mean of the gain	stddev
Microsoft	16. Jun 99	10.12.1999	18.74 %	10.13 %	13.06
Apple	16. Jun 99	10.12.1999	122.10 %	51.97 %	28.58
Adobe	16. Jun 99	10.12.1999	74.75 %	34.55 %	24.72
Boeing	23. Jun 99	17.12.1999	-9.56 %	-4.34 %	10.29
General Motors	23. Jun 99	17.12.1999	14.89 %	7.99 %	11.50
AT & T	25. Jun 99	21.12.1999	-0.75 %	0.72 %	14.14
Coca - Cola	25. Jun 99	21.12.1999	-6.93 %	-2.54 %	12.23
Walt Disney	25. Jun 99	21.12.1999	-3.58 %	-0.66 %	12.40
Hewlett Packard	25. Jun 99	21.12.1999	19.55 %	11.41 %	19.51
McDonalds	25. Jun 99	21.12.1999	4.30 %	2.55 %	10.67

Table 2.1: simulation with a random model compared to the buy & hold strategy

The simulation in table 2.1 shows that random models have a gain of about half that of the *buy & hold* strategy. This can be explained:

Let $p(i)$ be the ratio between the opening price of *day* i and the opening price of *day* $(i-1)$. With this ratio we can compute the gain factor that results by the simulation over N days with the *buy & hold* strategy:

$$gain = \prod_{i=1}^N p(i) = \frac{open(1) \cdot open(2) \cdot \dots \cdot open(N)}{open(0) \cdot open(1) \cdot \dots \cdot open(N-1)} = \frac{open(N)}{open(0)}$$

Let S be a set that contains all the $p(i)$: $S = \{ p(1), p(2), \dots, p(N) \}$. A random model chooses a random subset of set S . We assume that all $p(i)$ are the same:

$$p(i) = 1 + p$$

where p is the average percentage gain per day with the *buy & hold* strategy. Note, the expected value of the gain is also depending on the distribution of the $p(i)$. The assumption that all the $p(i)$ are the same is a simplification.

The expected value of the gain factor produced by a random model gives (any combination of elements of the set possible):

$$E(\text{gain}) = \frac{1}{2^N} \sum_{i=0}^N \binom{N}{i} \cdot (1+p)^i = \left(1 + \frac{p}{2}\right)^N$$

We can also arrive at the same result in a different way:
The gain factor $g(i)$ at a certain day for a random model is

$$g(i) = \begin{cases} 1 & \text{with probability } 0.5 \text{ (decision was sell, state is MONEY)} \\ p(i) & \text{with probability } 0.5 \text{ (decision was buy, state is STOCKS)} \end{cases}$$

The expected gain factor of a random model at a particular day is

$$E(g(i)) = 0.5 \cdot 1 + 0.5 \cdot p(i) = 0.5 \cdot (1 + p(i)) = 0.5 \cdot (1 + (1+p)) = 1 + p/2$$

From that, we can see that the expected value of the gain with a random model at a certain day is just half that of p , which is the average percentage gain per day with the *buy & hold* strategy. The expected gain for a random model is

$$E(\text{gain}) = \prod_{i=1}^N E(g(i)) = \left(1 + \frac{p}{2}\right)^N$$

Example: *Apple, 125 days, buy & hold: 122.1%*
 $p = (1+1.221)^{(1/125)} - 1 = 0.00640$ (+0.64% per day)
 $E(\text{gain}) = 1.49125$ (= +49.125%)
simulation: +51.97%

More importantly than the mean, the gain produced by random models typically has a very high standard deviation; this makes it very hard to show that a given (presumably not random) model is not just good on the test-set by coincidence.

One way to get trust into the model, is to repeat the simulation with a new, disjoint test-period. But with only two simulations, we still can not trust the model yet. We would have to make a lot of simulations, always with disjoint test-sets. But we will not have enough disjoint test-sets as one single test-set contains data for half a year.

Another way to get trust into the model is to analyse the transactions done by the model. From the transaction-sequence (which is stored in a log-file), we can compute the mean and standard deviation from the single transaction gains. If we have a small mean and a huge standard deviation within these transactions, it is a sign that our gain was basically caused by coincidence: Already only a few more similar transactions could change the good result completely. On the other hand, having a rather high mean and a small standard deviation, our gain was more or less constantly growing over the time, which would be a very good sign.

Furthermore, we can also judge a model by the ratio between good and bad decisions. A good decision is to buy at a price lower than the opening price of the following day and to sell at a higher price than the opening price of the following day. A bad decision is to sell at a lower price than the opening price of the following day and to buy at a higher price than the opening price of the following day. A good model is supposed to make more good than bad decisions, even though we are not optimizing the model for making good decisions. Although this ratio does not say anything about how good and how bad the decisions were, it gives us a hint, whether we can trust a model or not.

And finally, from the evolution of the optimal model parameters over the simulation time (stored in the *paramX.txt* log-file of the according stock), we can make some conclusions about the model.

Assume, we have K training days and we have a period of $K+1$ days with constant optimal parameters. In that case, after the first K days of constant parameter, we always made the maximum possible gain as long as we have this constant parameter. A model whose evolution curve is constant over long periods might be very good. Note, the less constant the evolution curve was, the more random the decisions were.

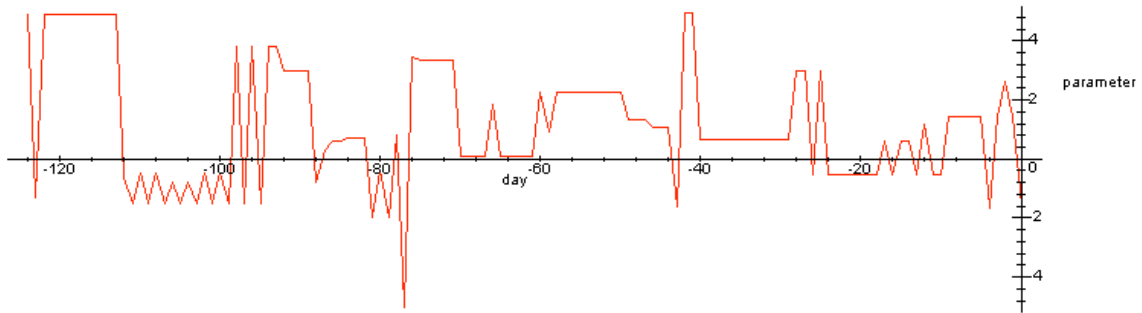


Figure 2.4: evolution of the optimal parameter over the simulation time. Model 1, Adobe, 25 training days.

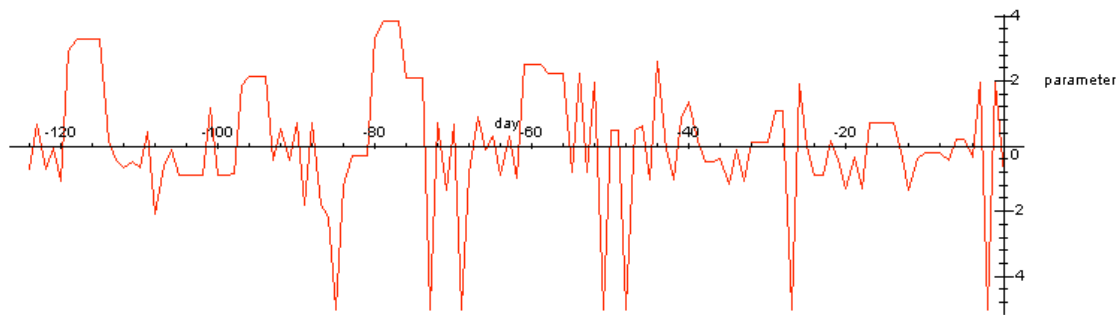


Figure 2.5: evolution of the optimal parameter over the simulation time. Disney, 5 training days.

We cannot compare two evolution curves with two different number of training days, as evolution curves with more training days are supposed to be much smoother, because moving a long training window for one day will not change very much in the optimal decision sequence. Therefore, we should only compare evolutions curves with the same number of training days.

3. LINEAR MODELS

In this chapter, we are looking at linear models in some more detail. Applied to our prediction problem, the general description of a linear models is

$$y = b_0 + \sum_{i=1}^N x_i \cdot b_i(\text{day})$$

where y could be either a decision or a price; x_i are the parameters to optimize and $b_i(\text{day})$ are numerical values depending on the day. The values for $b_i(\text{day})$ are not necessarily linear.

3.1 Linear decision models

Linear decision-models are models with a policy that gives a decision - BUY or SELL - for every day. The policies of linear decision models can be described as follows (for simplicity reasons only 2 parameters):

$$\text{decision}(\text{day}) = x \cdot A(\text{day}) + y \cdot B(\text{day}) + C(\text{day})$$

where x and y are the parameters that are going to be optimized, and $A(\text{day})$, $B(\text{day})$ and $C(\text{day})$ are numerical values depending on the day, e.g. $A(\text{day}) = \text{open}(\text{day}) \cdot \text{close}(\text{day}-1)$

If $\text{decision}(\text{day})$ is greater or equal to zero, our decision is BUY, otherwise SELL.

3.1.1 Relationship between linear decision models and perceptrons

Finding the optimal parameters for a linear decision model is equivalent to determine the weights of a perceptron (type of neural network):

$$\text{decision} = p_0 \cdot A + p_1 \cdot B + C$$

if ($\text{decision} \geq 0$) then BUY else SELL

where A,B,C are our data points and p_0 , p_1 our parameters.

We can write above equation also the following way (assume $C > 0$):

$$\text{BUY,} \quad \text{if} \quad p_0 \cdot \frac{A}{C} + p_1 \cdot \frac{B}{C} + 1 \geq 0 \quad \text{else SELL}$$

or in a different way:

$$\frac{B}{C} \geq -\frac{A}{C} \cdot \frac{p_0}{p_1} - \frac{1}{p_1}$$

drawing the data points into the xy-plane, we have to find a straight line such that this straight line (which is depending on the parameters p_0 and p_1) is separating the data points (on one side of the straight line only BUY-points, on the other side of the straight line only SELL-points):

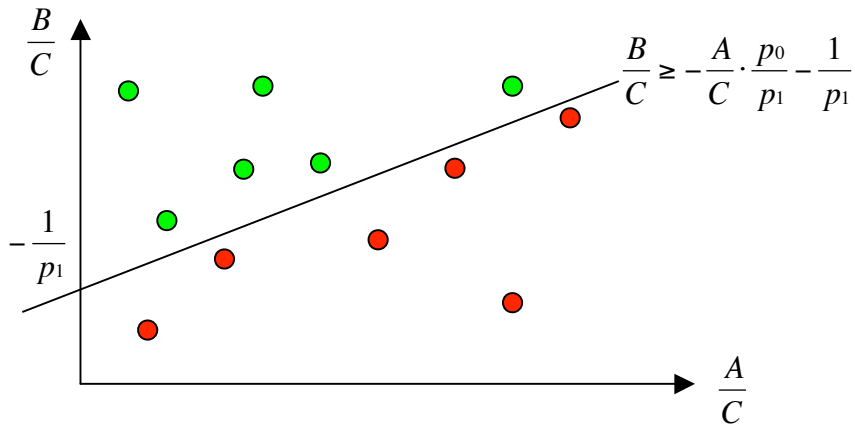


Figure 3.1: data points in the xy -plane. Green points: data points where the best decision is SELL . Red points: data points where the best decisions is BUY.

For every training day, we get either a red point or a green point. The maximization process is trying fit a straight line through the plane such that the red points and the green points are separated. If the maximization process manages to separate all the points, we reach the maximum possible gain in the training period.

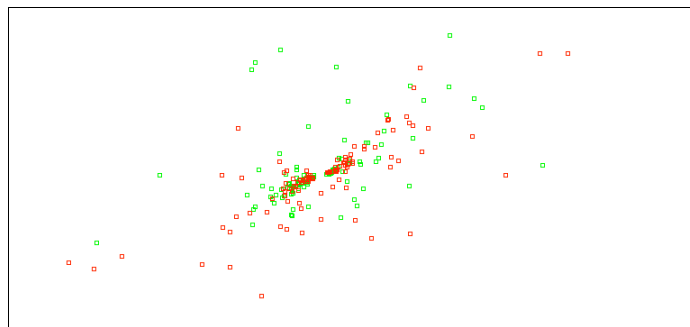


Figure 3.2: 200 data points of a decision model

3.1.2 Upper limit for the number of plateaus for linear decision - models

To estimate the upper limit for the number of plateaus, we describe the decision at $day\ k$ as a straight line in the xy -plane:

BUY, if (decision) ≥ 0 (assume $B(day) > 0$):

$$y \geq -\frac{A(day)}{B(day)} \cdot x - \frac{C(day)}{B(day)}$$

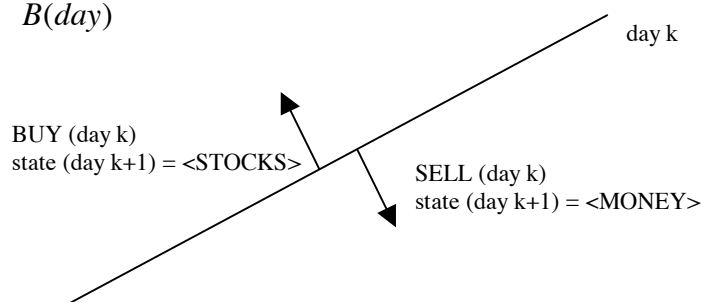


Figure 3.3: different states after 1 day

The straight line is cutting the xy -plane into 2 halves and therefore into 2 plateaus.
 For every day of the training period, we get a different straight line, as in general $A(\text{day})$, $B(\text{day})$ and $C(\text{day})$ will be different for every day.
 Looking at several days, we can see that in general, a new straight line is crossing all the existing straight lines:

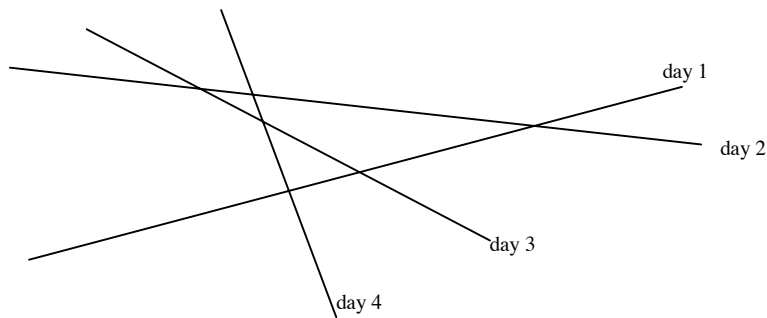


Figure 3.4: different plateaus after 4 days

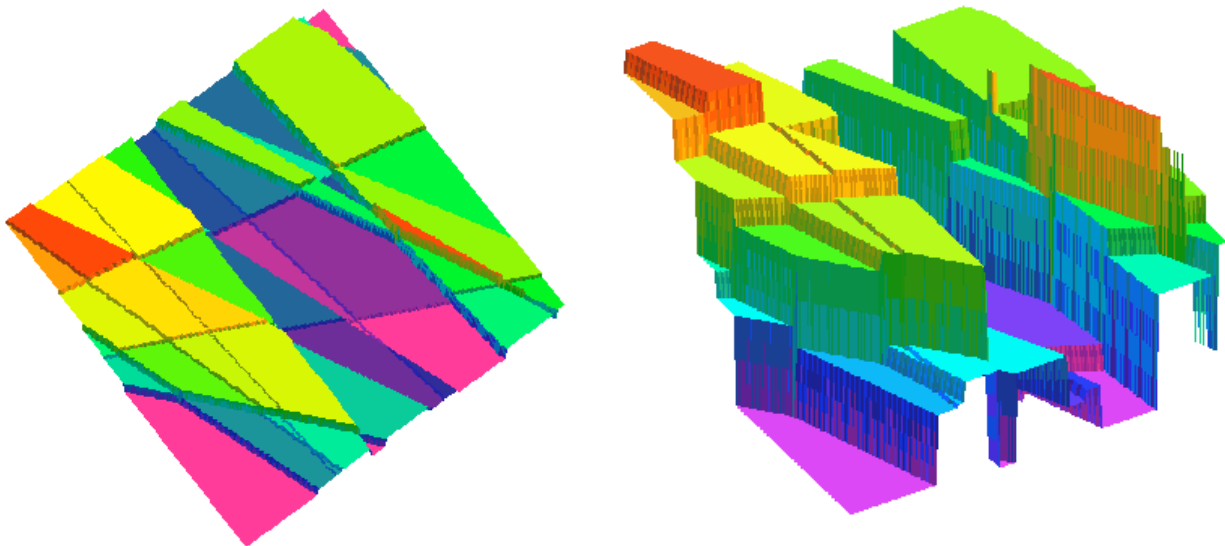


Figure 3.5: score function of a linear decision model

All the (convex!) sub-areas in the xy -plane contain parameters, that lead to the same decision sequence. To estimate the upper limit for plateaus, we have to count the sub-areas.

As every new straight line can cross at most all the existing lines, we can set up a recursive formula for the upper limit for the number of plateaus $P(N)$:

$$P(N) = \begin{cases} 2, & \text{if } N=1 \\ P(N-1) + N, & \text{if } N>1 \end{cases}$$

Solving this recursion for $P(N)$ gives:

$$P(N) = \frac{N \cdot (N + 1)}{2} + 1$$

where N is the number of lines (equal to the number of training days).

We can see that the upper-limit of plateaus for linear decision models is $O(N^2)$.

Note that, in the special case where we do not have a constant (C or $C(\text{day})$), all the lines are going through the origin:

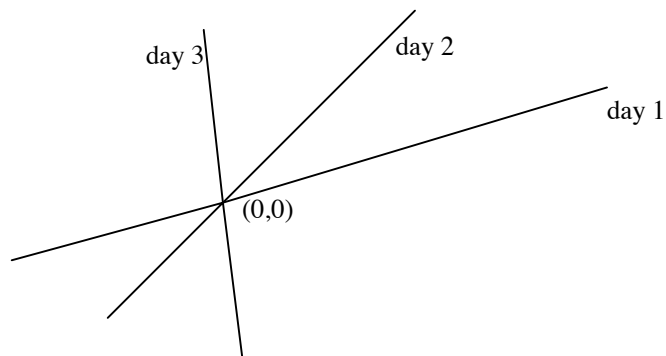


Figure 3.6: score function of a linear decision model without constant

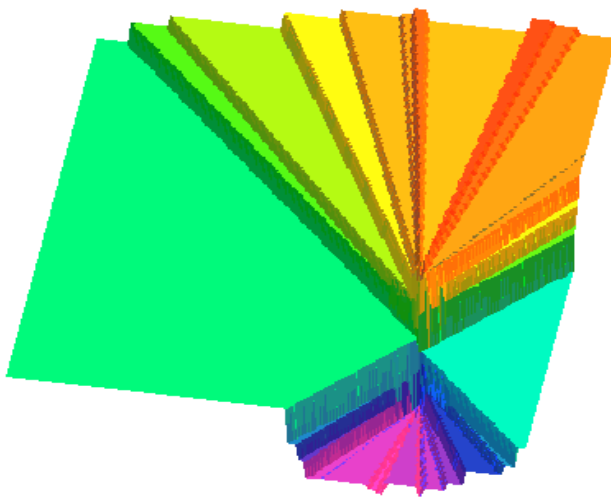


Figure 3.7 score function of a linear decision model without constant

In this special case there are only $2 \cdot N$ plateaus. The plateau with the maximum score can easily be found by walking around any circle with centre $(0,0)$. (Binary search for the boundaries)

3.2 Linear one-price models

The policy of **one-price models** can be described as follows (for simplicity reasons only two parameters x,y):

$$\text{actionPrice}(\text{day}) = x \cdot A(\text{day}) + y \cdot B(\text{day}) + C(\text{day})$$

$$\text{buyPrice}(\text{day}) = \min(\text{open}(\text{day}), \text{actionPrice}(\text{day}))$$

$$\text{sellPrice}(\text{day}) = \max(\text{open}(\text{day}), \text{actionPrice}(\text{day}))$$

3.2.1 Upper limit for the number of plateaus for linear one-price models

With this type of model, we can also describe the decision at any day by lines in the xy-plane. Here, we do not only get one single straight line per day, but many, parallel lines as we have many possible prices to buy or sell every day. Even though we are making the same decision every day, we will have different scores for two parameters as soon as one parameter leads to slightly higher or deeper price at any single day of the training.

For every day, we have $\lceil 16 \cdot (\text{open}(\text{day}) - \text{low}(\text{day})) \rceil$ BUY-lines, as we are only going to buy in the price-range between the opening price and the lowest price of the day. And every different price within this range will lead to a different score and therefore a different plateau. All the prices are rounded to $1/16$.

Analytical description of these lines:

Description of BUY-line i , $i \in [0, \lceil 16 \cdot (\text{open}(\text{day}) - \text{low}(\text{day})) \rceil]$, assume $B(\text{day}) > 0$

$$y \geq -\frac{A(\text{day})}{B(\text{day})} \cdot x + \frac{\text{low}(\text{day}) + \frac{i}{16} - C(\text{day})}{B(\text{day})}$$

analogous the description of SELL-line i , $i \in [0, \lceil 16 \cdot (\text{high}(\text{day}) - \text{open}(\text{day})) \rceil]$, assume $B(\text{day}) > 0$

$$y < -\frac{A(\text{day})}{B(\text{day})} \cdot x + \frac{\text{high}(\text{day}) - \frac{i}{16} - C(\text{day})}{B(\text{day})}$$

From the description of the lines, we can see that all these lines must be parallel.

As for the first day, we have only one state, which is $\langle \text{MONEY} \rangle$ and therefore we can only have BUY-lines:

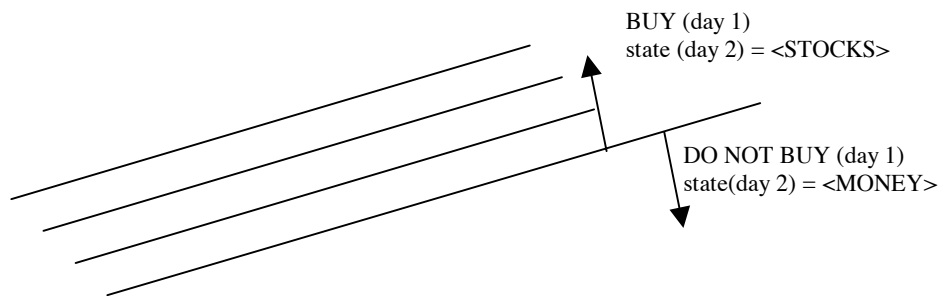


Figure 3.8: plateaus and states after the first day of simulation with a one-price model

For the second day, we also need to consider our state: $\langle MONEY \rangle$ or $\langle STOCKS \rangle$

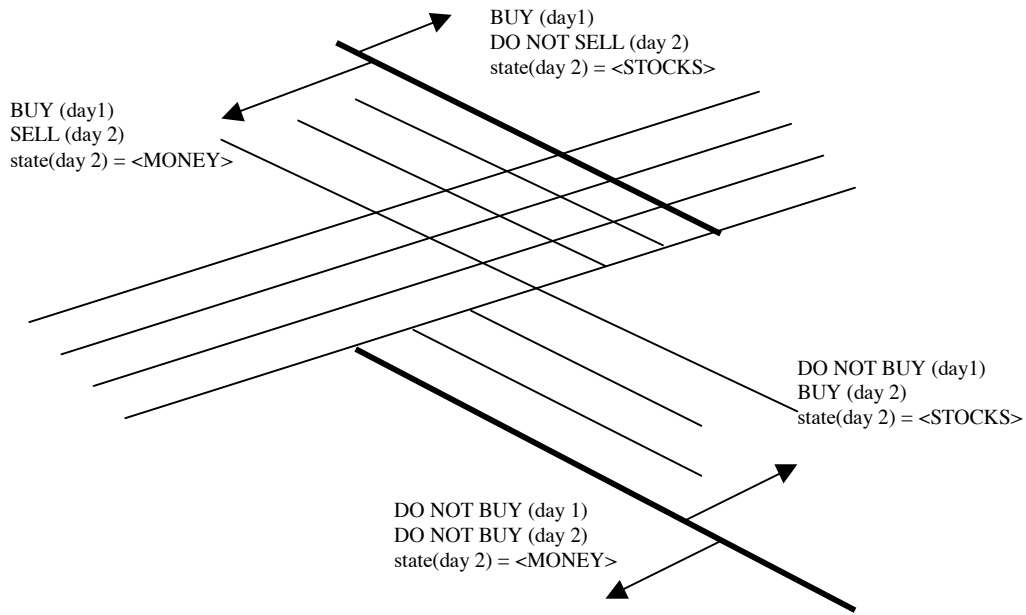


Figure 3.10: plateaus and states after the second day of simulation with a one-price model

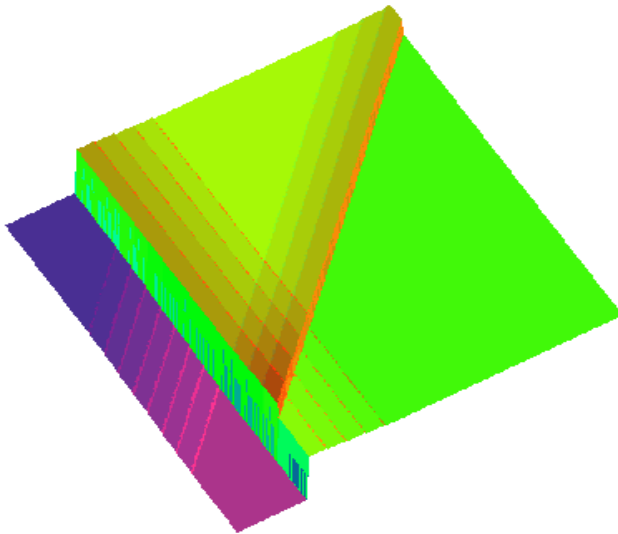


Figure 3.11: plateaus and states after the second day of simulation with a one-price model

We can estimate the upper limit for the number plateaus by counting the sub-areas. In the worst case, every line of $day\ k$ could cross all the lines of all days $< k$. But, as all the lines of a day are parallel, they do not cross each other.

Per day, we have $\lceil 16 \cdot (high(day) - low(day)) \rceil$ parallel lines. But these lines are not going through the whole xy -plane, because we are selling at least for the opening price and we are buying at most for the opening price. The only straight line that is going through the plane is the opening-price - line. However, in the worst case, the opening price is always the same as the lowest price (or the highest price) and in that case, all the lines are going through the whole plane.

As one line at *day k* can cross all the other lines from *days < k*, we can set up the recursion for the upper limit for the number of plateaus:

For simplicity reasons, let us assume that we have 40 lines per day (which means that we can buy or sell at 40 different prices every day).

lines(n) : number of lines in the *xy*-plane after day *n*
 $lines(n) = 40 \cdot n$

$\Delta plateaus(n)$: maximum increase of the number of plateaus at day *n*
 $\Delta plateaus(n) = (40+1) \cdot (lines(n-1)+1)$
 $= 41 \cdot (40(n-1)+1)$

plateaus(n): maximum number of plateaus after day *n*
 $plateaus(0) = 0$

$$plateaus(N) = plateaus(0) + \sum_{k=1}^N \Delta plateaus(k)$$

$$plateaus(N) = \sum_{k=1}^N 41 \cdot (40(k-1) + 1)$$

simplifying this equation, we get

$$plateaus(N) = \frac{41}{2} (N(41N - 39))$$

Also in this case, the upper limit for the number of plateaus is $O(N^2)$. But this time we have a really huge constant factor!

3.2.2 Avoiding line segments

For the *Adaptive Squares* maximization method (4.2.2), we need to have a guarantee that there are not line segments in the *xy*-plane.

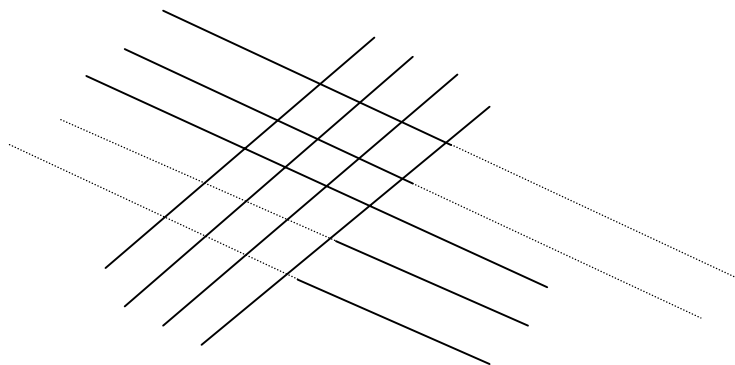


Figure 3.12: continuous lines in the *xy*-plane

We can avoid line segments by adding very small noises to the buy and sell price of the current day:

$$buyprice(day) = buyprice16(day) + noise(day) + \varepsilon \cdot \min(high(day), sellprice16(day))$$

$$sellprice(day) = sellprice16(day) + noise(day) + \varepsilon \cdot \max(low(day), buyprice16(day))$$

where $buyprice16(day)$ and $sellprice16(day)$ are prices which are rounded to $1/16$; ε is a very small noise.

3.3 Linear buyprice/sellprice models

Buy/sellprice models are different in a way that we have two, maybe completely different policies: One policy for the buyprice and one policy for the sellprice. In general, we can describe these linear policies as follows:

$$\text{buyPrice}(\text{day}) = \min(\text{open}(\text{day}), x \cdot A(\text{day}) + y \cdot B(\text{day}) + C(\text{day}))$$

$$\text{sellPrice}(\text{day}) = \max(\text{open}(\text{day}), x \cdot D(\text{day}) + y \cdot E(\text{day}) + F(\text{day}))$$

3.3.1 upper limit for the number of plateaus for linear buy/sellprice models

The upper limit for the number of plateaus for N days still remains the same as for linear one-price models $O(N^2)$. But the lines are just going to be parallel for the same state (MONEY or STOCKS), as we have two different linear policies for the two states.

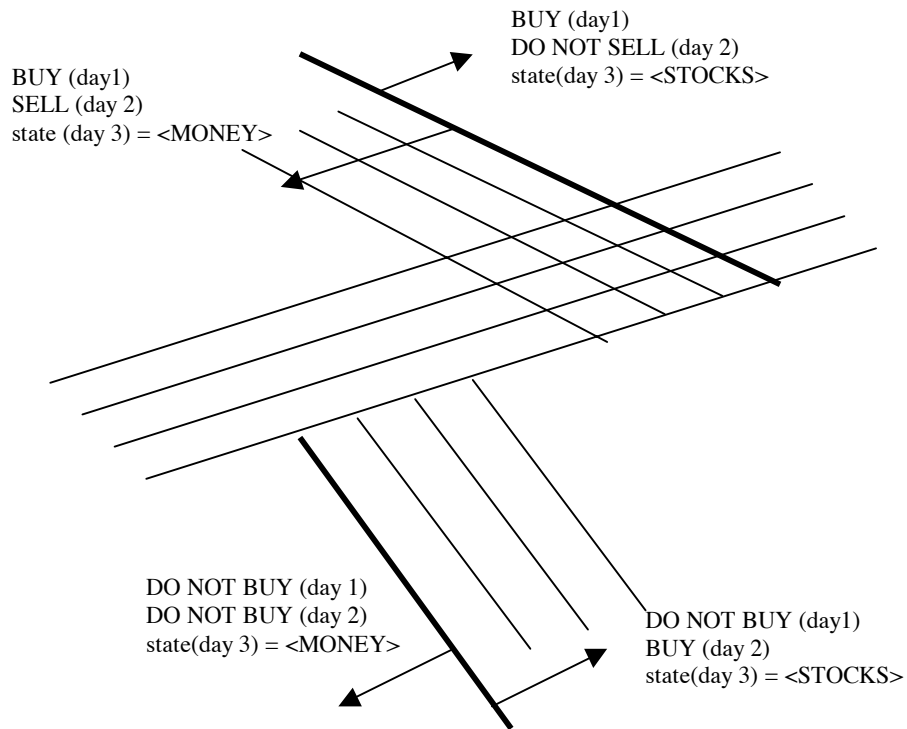


Figure 3.13: plateaus and states after the second day of simulation with linear buy/sellprice model

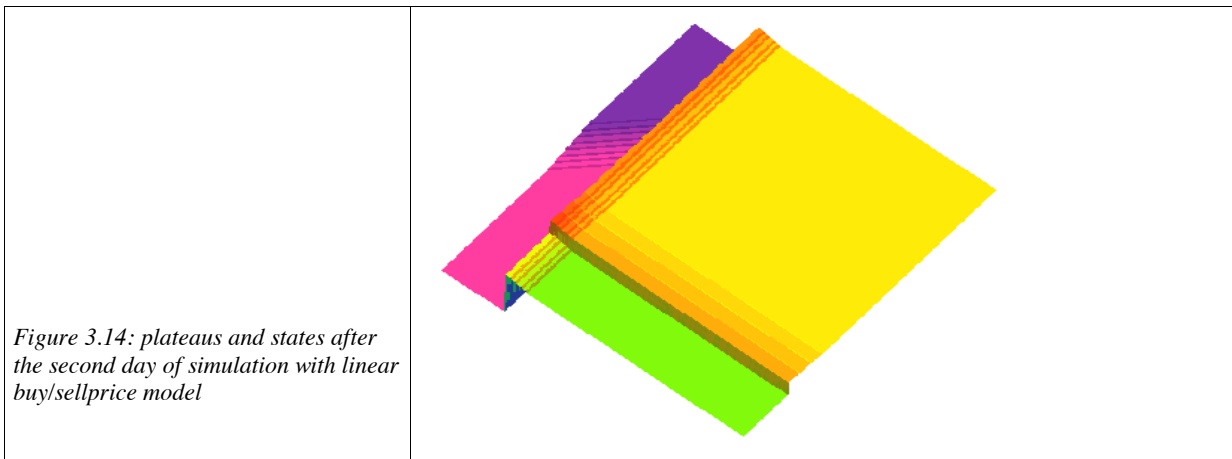


Figure 3.14: plateaus and states after the second day of simulation with linear buy/sellprice model

3.3.2 Special case: policies with independent parameters

In the case of policies with independent parameters, the buy-price at any day is just depending on x and the sell-price is just depending on y :

$$\begin{aligned} \text{buyprice}(\text{day}) &= \min(\text{open}(\text{day}), x \cdot A(\text{day}) + B(\text{day})) \\ \text{sellprice}(\text{day}) &= \max(\text{open}(\text{day}), y \cdot C(\text{day}) + D(\text{day})) \end{aligned}$$

Description of the lines:

$$\begin{aligned} i &\in [0, \lceil 16 \cdot (\text{open}(\text{day}) - \text{low}(\text{day})) \rceil], \text{ assume } A(\text{day}) > 0 \\ j &\in [0, \lceil 16 \cdot (\text{high}(\text{day}) - \text{open}(\text{day})) \rceil], \text{ assume } C(\text{day}) > 0 \end{aligned}$$

$$x \geq \frac{\text{low}(\text{day}) + \frac{i}{16} - B(\text{day})}{A(\text{day})}$$

$$y < \frac{\text{high}(\text{day}) - \frac{j}{16} - D(\text{day})}{C(\text{day})}$$

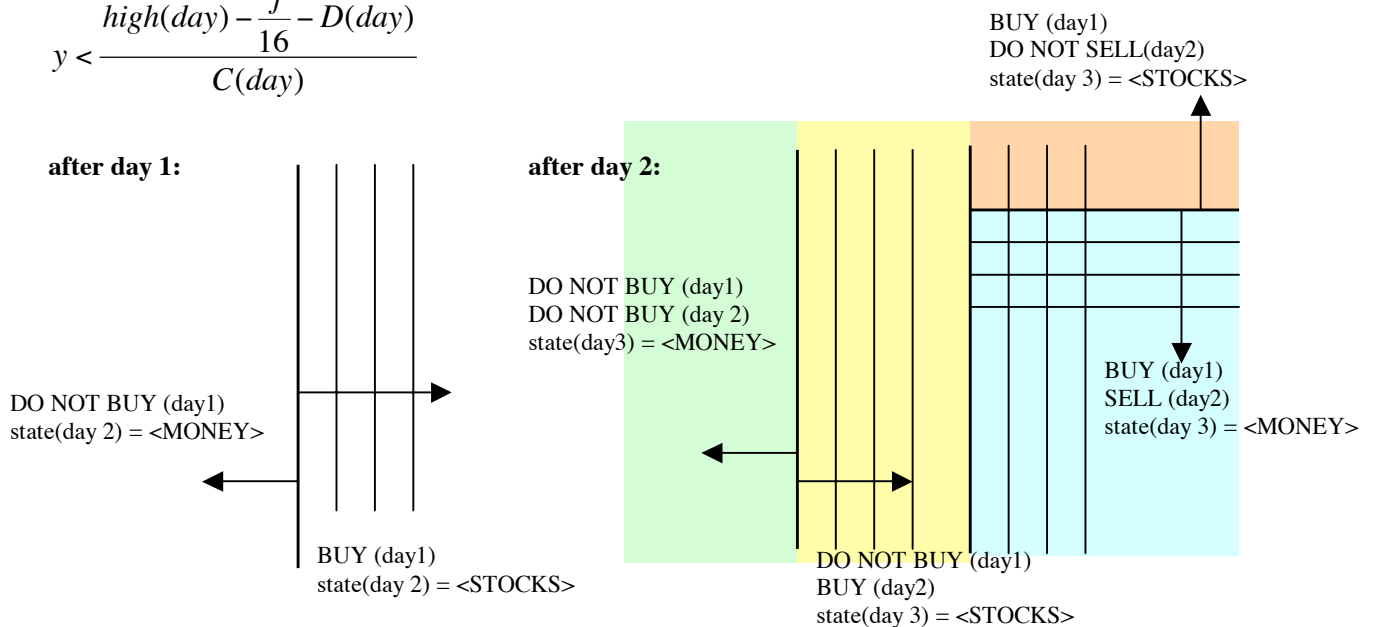


Figure 3.15: plateaus and states of models where the parameters x and y are independent.

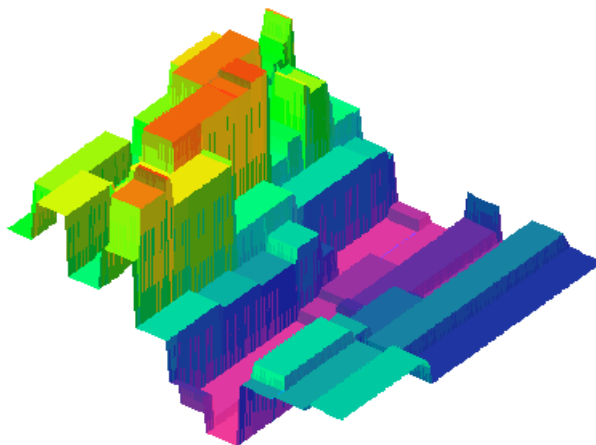


Figure 3.16: The score function of a buyprice/sellprice model where x and y are independent

3.3.3 Avoiding line segments

To avoid line segments in buyprice/sellprice models, we have to consider also the fact that the BUY-lines and the SELL-lines are not parallel to each other any more:

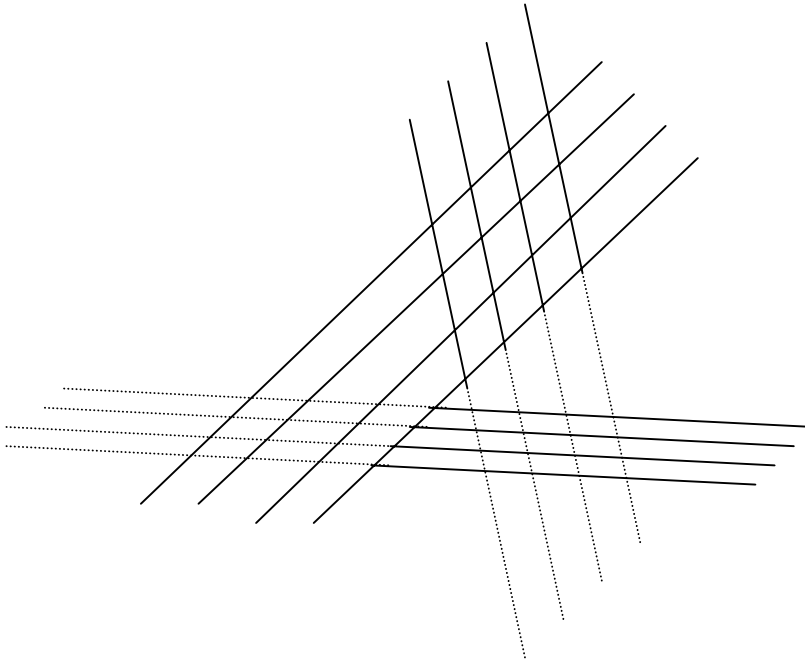


Figure 3.17 continuous lines in buy/sellprice models

Contrary to linear one-price models, we have to add a small noise to the score even though we are not buying or selling. The small changes in the case of DO_NOTHING can be described as follows:

$$\begin{aligned} money_{day} &= money_{day-1} + \varepsilon \cdot \min(high(day), sellprice16(day)) \\ stocks_{day} &= stocks_{day-1} + \varepsilon \cdot \max(low(day), buyprice16(day)) \end{aligned}$$

The policies for the sellprice and the buyprice of the day remain the same as in the case of one-price models:

$$\begin{aligned} buyprice(day) &= buyprice16(day) + noise(day) + \varepsilon \cdot \min(high(day), sellprice16(day)) \\ sellprice(day) &= sellprice16(day) + noise(day) + \varepsilon \cdot \max(low(day), buyprice16(day)) \end{aligned}$$

where buyprice16(day) and sellprice16(day) are prices which are rounded to 1/16; ε is a very small noise factor.

4. MAXIMIZATION METHODS

In this chapter, we are studying maximization methods for a special kind of functions, which are the result of a prediction process for example.

$$f: \mathbb{R}^N \rightarrow \mathbb{R}, \quad (p_0, p_1, p_2, \dots, p_{N-1}) \mapsto f(p_0, p_1, p_2, \dots, p_{N-1})$$

where p_0, p_1, \dots, p_{N-1} are continuous parameters. The function f is piecewise constant on the parameters, which means that the gradient at any point of the function is either 0 or $\pm\infty$. That's the reason why all the efficient standard maximization methods fail.

We assume that every plateau (range of parameter space with constant function value) is unique. Under the assumption of unique plateaus, we know that whenever two different arguments lead to exactly the same function value, there can not be any other plateau between these two arguments.

Figure 4.1 shows such functions in two dimension:

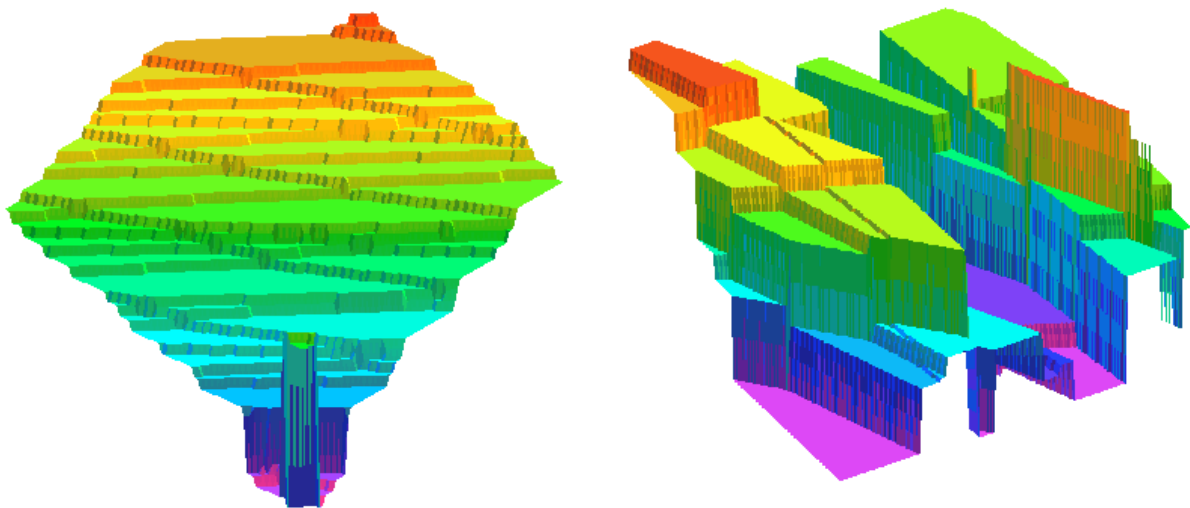


Figure 4.1: piecewise constant functions in two dimensions

A property of these functions is that we will never have a guarantee that we have found the global maximum unless we know the function value of every single plateau. Unfortunately, such functions can have so many plateaus, that we are not able to visit every single plateau within a reasonable amount of time (depends on computational resources).

For functions in one dimension, there is an efficient algorithm known which finds the global maximum in $O(P)$, where P is the number of plateaus.

For functions whose plateaus are caused by straight lines in two dimensions (Figure 4.1) we have developed an efficient algorithm that is also able to find the global maximum. We also propose an extension of this algorithm to any dimension.

If P is too large or the plateaus were not caused by straight lines in two dimensions or hyper-planes in any dimension, we have developed an heuristic algorithm based on spirals in two dimensions and hyper-spheres in n -dimensions that turned out to be very fast and reliable.

We are only discussing the maximization problem. The minimization problem can be solved by maximizing the same function with negative sign.

4.1 Maximization in one dimension

For functions in one dimension, there is an efficient algorithm known, which is working in $O(P)$, where P is the number of plateaus. The basic idea of this algorithm is to find all boundaries of all the plateaus by binary search.

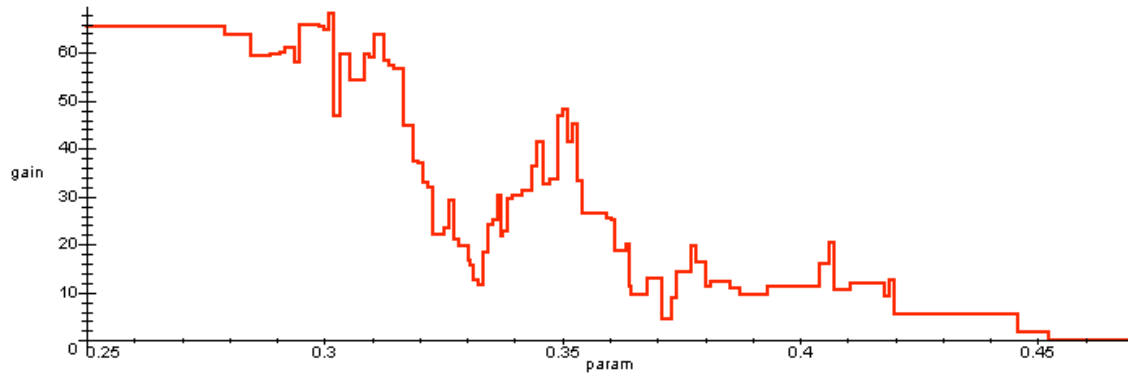


Figure 4.2: a piecewise constant function in one dimension

For the correctness of this algorithm, we need to have a guarantee that every single plateau is unique in its function value. Under this assumption, finding exactly the same value for two different arguments, we can be sure that there is no other plateau between these two arguments. Therefore we are able to search for the plateau - boundaries by binary search. We assume that we have a defined parameter range with a leftmost point and a rightmost point.

The algorithm itself is quite simple: Starting from the leftmost argument, we are looking for the boundary of the first plateau next to our leftmost argument by binary search. Step by step, we set the leftmost point onto the next plateau and restart the binary search for the next plateau boundary. As soon as we set the leftmost point onto the same plateau as the rightmost point, we can stop.

4.1.1 Complexity analysis

Binary search for one plateau-boundary:

$$T(1, \varepsilon) = \log_2(1/\varepsilon)$$

where ε is the precision of the binary search (e.g. machine epsilon).

We are repeating the binary search for the boundaries of every plateau. P plateaus have $(P-1)$ boundaries:

$$T(P, \varepsilon) = (P-1) \cdot \log_2(1/\varepsilon)$$

That means that we have an algorithm that is working in $O(P)$, which is optimal.

However, we have to be very careful, as the function might consist of a very huge number of plateaus, and $\log_2(1/\varepsilon)$ can be a rather huge constant factor (e.g. $\log_2(1/\varepsilon)=64$ using double precision).

4.2 Maximization in 2 dimensions

4.2.1 Spiral - method

The *Spiral method* is a heuristic maximization method based on spirals, which turned out to converge very quickly into a (local) maximum.

Starting from the outside of the spiral, we are walking along the spiral towards the spiral centre, which is always our currently best argument. In every turn around the spiral, we are evaluating the function exactly at one random point. As soon as we find a function value that is higher than the value of the current spiral centre, we stop with the current spiral and start with a new spiral around the new, currently best point as centre of the spiral. We choose the radius of the new spiral as twice the distance from the old to the new spiral centre.

As soon as we get several times the same value as the spiral centre one after another, we are in a local maximum with a high probability. We repeat the same procedure several times with different random start points.

We will never have a guarantee that we have found the global maximum. Nevertheless this method turned out to find the global maximum often already after a very short time compared to other maximization strategies.

4.2.1.1 Choosing random start points

We are not just choosing any random point and radius to start with a spiral, as this might be rather bad, because the initial radius could be too small. Therefore we analyse the function at first on a random direction through a real random start point. Along this direction, we are looking for the leftmost and the rightmost discontinuity of the function by binary search. We choose our start point as the middle of these two discontinuities in the function. We also guarantee that the initial radius is larger than the distance from the start point to the leftmost point.

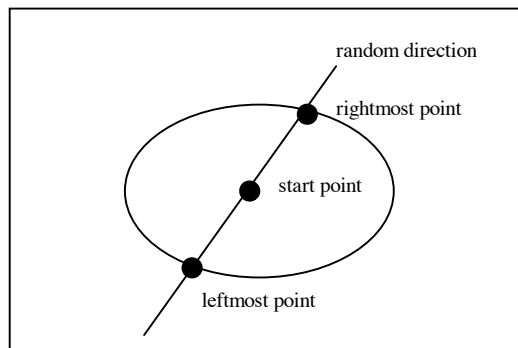


Figure 4.3: choice of a random start point

4.2.1.2 Properties of the spirals

The closer we get to the centre of the spiral, the more function evaluations we want to make. Therefore, halving the radius of the spiral, we always evaluate the function at a constant number of points (K). The point $p(n)$ which we are evaluating in the n -th turnaround can be determined by the radius $r(n)$ and by a "random" angle $\omega(n)$.

$$r(n) = r_0 \cdot \left(\frac{1}{2}\right)^{\frac{n}{K}}$$

$$\omega(n) = 2\pi(n\phi - \text{floor}(n\phi))$$

$$\bar{p}(n) = r(n) \cdot \begin{pmatrix} \cos(\omega(n)) \\ \sin(\omega(n)) \end{pmatrix}$$

where

ϕ golden ratio (0.618034)
 K constant number of function evaluations
 r_0 initial radius of the spiral
 n n -th turnaround

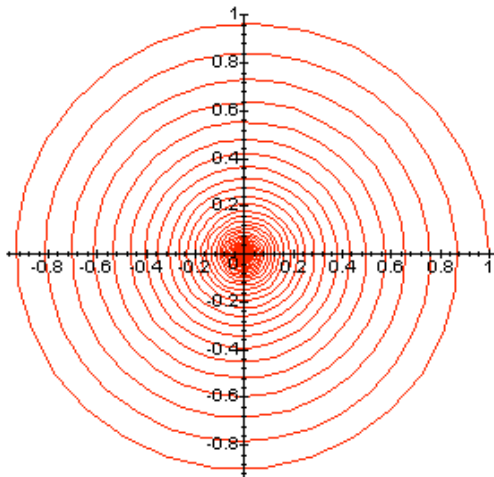


Figure 4.4: spiral with $K=5$

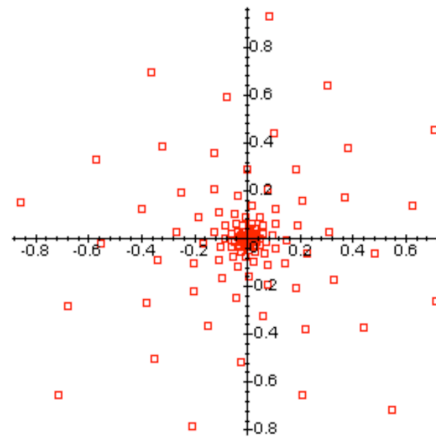


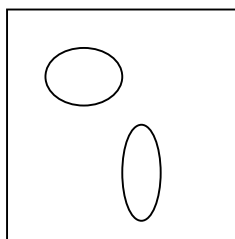
Figure 4.5: evaluated points on the spiral

4.2.2 Adaptive Squares method

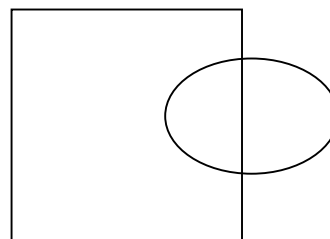
This maximization method is a special maximization method for two dimensional functions, that result by always separating the xy -plane with straight lines. Therefore, it follows that all the plateaus are convex. Such functions could be score-functions in a prediction process with linear decision models for example.

The idea of the algorithm is to start with a single square that includes the whole (defined) parameter space. Whenever there is plateau completely hidden within a square, we are going to split this square into 4 sub-squares and apply the algorithm recursively to all the four sub-squares. With three rules, we are able to detect all the cases where we do not need to split the square

For the correctness of the algorithm, we need to exclude the following special cases:



a) islands

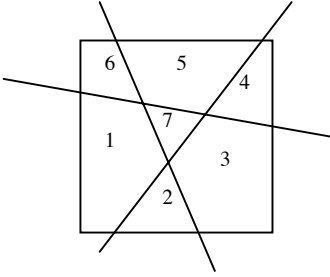
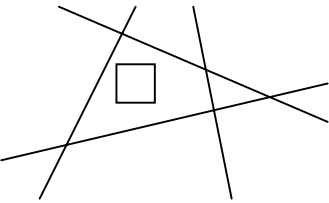
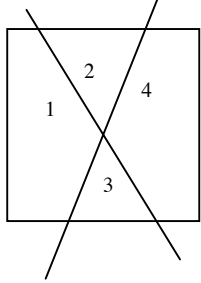
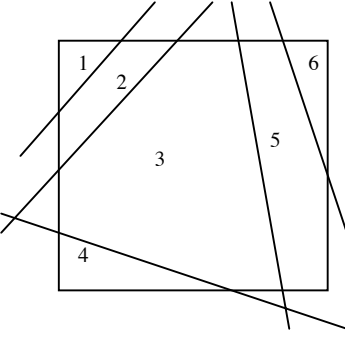


b) half-islands

Figure 4.6: special cases which we have to exclude: a) islands and b) half-islands

By the assumption that all the plateaus are caused by straight lines in the xy -plane, we do not need to bother about the special cases in figure 4.6.

4.2.2.1 Rules for splitting a square into 4 sub-squares:

<p><i>Two or more crossings of straight lines inside the square</i></p>	<p>As it is possible to hide a value (7) within the square, we need to split the square into 4 sub-squares.</p>	
<p><i>All 4 corners have the same score</i></p>	<p>With the guarantee of no islands and no half-islands, we can be sure that the whole square is contained within a plateau. And therefore, we do not need to split the square.</p>	
<p><i>4 or less different scores on the border</i></p>	<p>In this case, we can have at most one crossing of two lines within the square. To hide a plateau inside the square, there are at least two crossings inside the square necessary.</p> <p>As all the values inside the square also appear on the edges of the square, we do not need to split the square.</p>	
<p><i>many lines, but no crossings inside the square</i></p>	<p>Whenever there are no crossing of lines inside the square, all the values inside also appear on the edges of the square.</p> <p>To detect whether there is a crossing of lines inside the square or not, we need to have a look at all the edges of the square.</p> <p>We write all the values on the edges into a list, which we then try to resolve by our algorithm. If we manage to resolve this list, there is no crossing of any line inside the square, and therefore we do not need to split the square.</p>	

4.2.2.2 Algorithm to detect crossings of lines inside a square

Example (picture above):

- 1.) compute all the discontinuities of the function on all the edges and write the function values in order into a list.
The values on the edges can be found by binary search (see maximization in 1 dimension).
- 2.) if the list contains only 1 element:
return **true** if this element is a corner value,
return **false** if this element is not a corner value.
- 3.) find all the indexes of the list elements whose values are the same as List[0].
- 4.) if there is only one index (which is 0) and List[0] is not a corner value, then return **false** (which means that we can not resolve this list)
- 5.) create sub-lists between the indexes found
- 6.) recursively resolve all the sub-lists.
return **true** if all the sub-lists are resolvable, return **false** otherwise

List = [3,5,6,5,3,2,1,2,3,4]

List[0] = 3
List = [3,5,6,5,3,2,1,2,3,4]
indexes = [0,4,8]

sublist1 = List[1..3]=[5,6,5]
sublist2 = List[5..7]=[2,1,2]
sublist3 = List[8..8]=[4]

However, it is impossible to detect all the cases where there is no splitting of the square needed, because there exist cases which are not detectable just by analysing the values on the edges of the square:

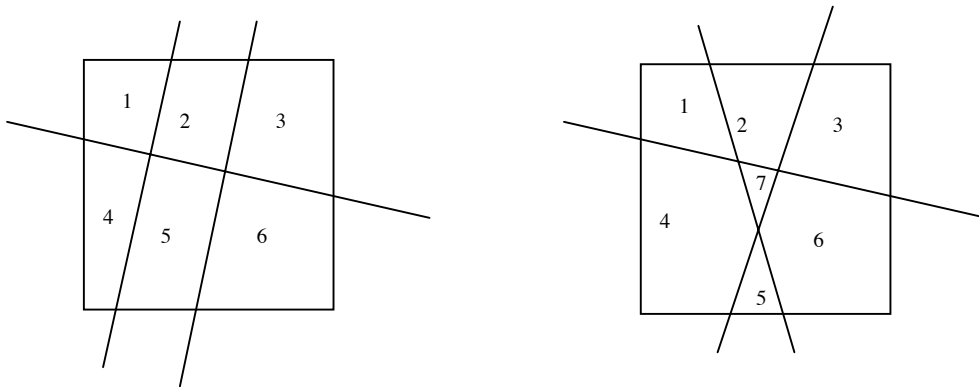


Figure 4.7: special case that can not be detected by analysing the values on the edges

In Figure 4.7, we have exactly the same sequence of values on the border ($List=[5,6,3,2,1,4]$). In the first case, we actually do not need to split the square into sub-squares, but with our rules, we are still splitting the square. But this is not a tragedy as we are never making a mistake, but sometimes we are splitting a square although we actually do not have to. There is another special case:

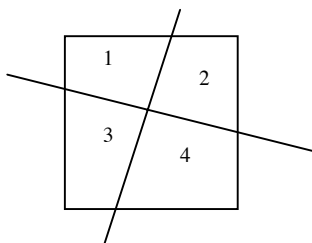


Figure 4.8: special case where we can not detect the crossing with our algorithm

In the case of figure 4.8, our algorithm to detect crossings would return true, which means that it was able to resolve the list ($List=[4,2,1,3]$), as the list only contains corner values. But this case is not a tragedy, because first of all, we do not need to split this square anyway and secondly, we will never call the function to resolve this list as we have only 4 different values on the border.

4.2.2.3 Complexity analysis

We assume that we have K straight lines in the xy -plane. Therefore we can have at most $O(K^2)$ plateaus.

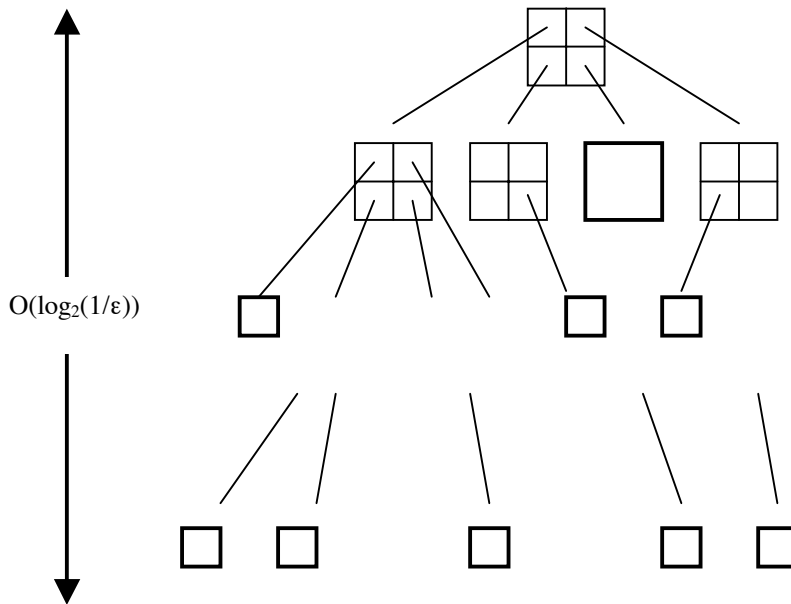


Figure 4.9: tree that results by the recursive splitting of the squares

maximum number of leaves in the tree

Assume, we have k straight lines in the xy -plane and M leaves in the tree. With every new line, we will cross all the existing k lines in the xy -plane in general and we can create at most $(k+1)$ new plateaus.

In the worst case, all these $(k+1)$ new plateaus lead to a further split of $(k+1)$ existing leaves. The recursion for the upper limit of leaves gives:

$leafs(K) \leq leafs(K-1) + (K+1)$. Solving this recursion, the maximum number of leaves is $O(K^2)$.

costs for one leaf of the tree

The costs to compute a single leaf are the costs that we have by the computation of all the values on all the four edges of the square. For that reason, we use the binary search method, discussed in the chapter about the maximization of functions in one dimension. As we have $O(K)$ lines, the costs for one leaf are at most $O(2 \cdot K \cdot \log_2(1/\epsilon))$. We have a factor of two because any line crosses the square twice. The costs for one leaf is $O(K \cdot \log_2(1/\epsilon))$.

cost for the nodes of the tree

The costs for any node in the tree is at most as high as the costs for its children and. Therefore the costs for any node is of the same order (but with constant factor) as the costs of a leaf.

The height of the tree can be at most $O(\log_2(1/\epsilon))$, because we can not split any square further than our precision allows. In the worst case, every node has only one child. Therefore, we get a factor of $O(\log_2(1/\epsilon))$ to the costs of the leaves.

Now, we are able to compute the overall costs:

$$O(K^2 \cdot K \cdot \log_2(1/\epsilon) \cdot \log_2(1/\epsilon)) = O(K^3 \cdot \log_2^2(1/\epsilon))$$

4.2.3 Comparison of the Adaptive Squares & Spiral method

A comparison of the two-dimensional maximization strategies showed that the *Spiral method* is excellent for finding a very good value very quickly. The comparison also proved that the *Adaptive Squares* method always found the global maximum with a high probability, as the *Spiral method* was never able to find a better value than the *Adaptive Squares* method in more than 200 different maximization runs (unless the maximum was not within the defined range). This is a clear hint, that our considerations about the *Adaptive Squares* method are correct.

To compare the two methods, we were optimizing the score-function of a linear decision model in stock market prediction. The tables with the complete result of the comparison can be found in the appendix. Here, there is only a short summary:

	adaptive squares method		spirals method (K=50, 100 restarts)	
	avg. max found (gain in [%])	avg. #evaluations	avg. max found (gain in [%])	avg. #evaluations
15 days	6.37505	108575	6.37449	40944
30 days	9.14927	439765	8.20685	43928
45 days	20.66483	1074787	19.49185	42930

Table 4.1: comparison of the adaptive squares method and the spiral method

In table 4.1, we can see that the number of function evaluations for the *Adaptive Squares* method is only increasing quadratic in the number of training days, although the worst case analysis gave $O(M^3)$ (M lines; for every day 1 line).

In general, we can say that the *Adaptive Squares* method can be used for functions that have a rather small amount of plateaus. But we can not give an exact number, as it completely depends on the time for one function evaluation, which is in our case a whole simulation of the stock market with a constant parameter for a model.

Even though the *Spiral method* did not find the global maximum in every case, it always found a very good value in a short time. The *Spiral method* is a good heuristic method to use in cases where there exist so many plateaus in the function that we cannot visit every single one. The number of function evaluations for the *Spiral method* can be determined by the choice of K and the number of restarts (should be adapted to the problem).

4.3 Maximization in n-dimension

4.3.1 Random directions

Starting in a random point, we are maximizing the function in one dimension along a random direction through that point. For that, we use the maximization method for functions in one dimension (see chapter 4.1). From the point with maximum value along this direction, we lay a new random direction and repeat the same procedure.

As soon as we are in local maximum, we will probably not be able to find any better value. After maximizing in several directions with no success, we assume that we are in a local maximum.

We choose several start-points and repeat the whole procedure in order to find the global maximum.

Unfortunately, this maximization method tends to be very slow, even though we are not maximizing on the whole random direction, but only within a certain, adaptive window - size. This method also tends to be rather unreliable to find the global maximum for functions that have very small plateaus; it is unlikely to hit a certain, very small plateau only with shooting into a random direction.

4.3.1.1 Generation of d-dimensional random directions

$d \leq 3$ Generate a vector of uniform distributed random variables. Repeat this until the norm of the generated random vector is less or equal to 1 (point must be within the sphere). Project the point onto the sphere by normalizing the vector.

$d > 3$ The higher the dimension, the more inefficient above method will become. Therefore, we generate random directions by normalizing a random vector of normal-distributed random variables. To generate normal-distributed random variables, we use the Box-Muller method, which is described in chapter 4.3.1.2.

4.3.1.2 Box-Muller method

Transformation of uniform distributed random variables into normal distributed random variables. With two uniform distributed random variables, we can generate two normal distributed random variables:

$$n_1 = \sqrt{-2 \cdot \ln(u_1)} \cdot \sin(2\pi \cdot u_2)$$

$$n_2 = \sqrt{-2 \cdot \ln(u_1)} \cdot \cos(2\pi \cdot u_2)$$

where

n_1, n_2 : normal distributed random variables

u_1, u_2 : uniform distributed random variables

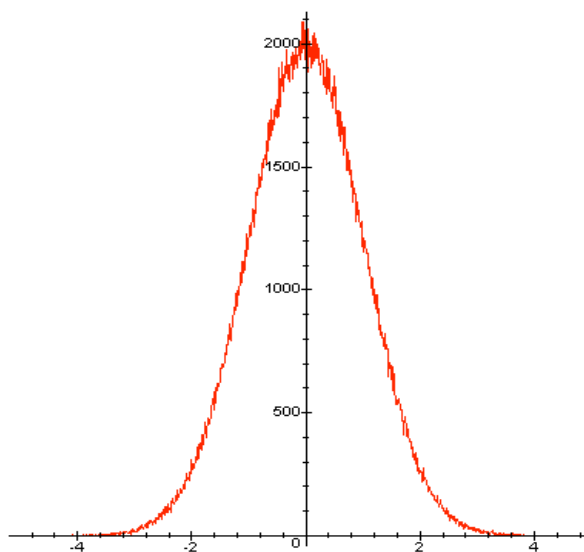


Figure 4.10: Test with normal distributed random variables generated by the Box-Muller method

4.3.2 Hyper-spheres method

We can extend the *Spiral-method* in two dimensions to any dimension by just a small change. Instead of evaluating points on a spiral around the current maximum, we are evaluating random points on d -dimensional hyper-spheres around the current maximum. In two dimensions, we evaluate one point per turn around the centre. In d dimensions, we evaluate the function at exactly 1 random point on the hyper-sphere of radius $r(n)$, where n is the n -th hyper-sphere around the centre. To keep this method analogous to the spiral-method, we also need to increase the constant number of evaluations (K) done per halving of the radius. K is depending on the dimension d :

$$K_d = K_2 \cdot 2^{d-2} \text{ for } d \geq 2$$

Reasonable values for K_2 turned out to be around 20-50, which means that every time we halve the distance to the centre of the spiral in two dimensions, we evaluate 20 random points. In that case, we would evaluate 40 points in 3 dimensions and 80 points in 4 dimensions, every time we halve the radius.

Now, we can compute the radius of the n -th hyper-sphere around the centre (current maximum):

$$r(n) = r_0 \cdot \left(\frac{1}{2}\right)^{\frac{n}{K_d}}$$

$$\vec{p}(n) = \vec{p}_0 + r(n) \cdot \text{randDir}_d$$

The point $\vec{p}(n)$ that we choose to evaluate on the hyper-sphere of radius $r(n)$ should be chosen by random as we want to avoid patterns in the choice of the evaluated points.

4.3.3 Hyper-cubes method

We believe that we can extend the *Adaptive Squares* method (2 dimension) to any dimension. However, this theorem has not been proved or implemented yet. The assumptions we have to make for the correctness of the algorithm remain the same as in two dimensions.

The idea is the same as the idea of the *Adaptive Squares* method: At first, we describe our m -dimensional parameter-space by a m -dimensional hypercube. Every time, a hypercube contains a plateau (volume in 3d) that we can not see on any face of the hypercube, we split the hypercube into 2^m sub-hypercubes.

The rules to split a hypercube can be derived from the two dimensional problem:

(m -dimensional hypercube)

all 2^m corners have the same value

We do not need to split the hypercube, under the assumption that we can not have any islands or half-islands within the hypercube.

we have less or equal $0.5(m(m+1)) + 1$ different values on the surface of the hyper-cube.

A hyper-cube of dimension m has $2^{m-1} + 2$ faces.

To completely hide a m -dimensional hyper-volume within a m -dimensional hyper-cube, there must be at least $(m+1)$ crossings of $(m-1)$ - dimensional hyper-planes.

That means, with only m crossings, it is impossible to hide a complete hyper-volume of dimension m .

m crossings inside the hyper-cube will produce $0.5(m(m+1)) + 1$ different plateaus on the faces.

Therefore if we have less or equal $0.5(m(m+1)) + 1$ different plateaus on the faces of the hyper-cube, we can be sure that we do not need to split.

no crossings of hyper-planes inside the hyper-cube

If there are no hyper-planes crossing at all inside the hyper-cube, we do not need to split the hyper-cube, as all the values inside appear on at least one face of the hypercube.

In three dimensions: if there are two planes crossing inside the cube, there is least on one face of the cube a crossing of straight lines. And we can detect this case by the 2-dimensional algorithm (by resolving a list).

In m -dimensions, we can reduce the problem of detecting crossings to a problem in $(m-1)$ dimensions: Every face of the m -dimensional hyper-cube is a hyper-cube of dimension $m-1$. A hyper-cube of dimension m has $(2^{m-1}+2)$ faces.

Recursively, we can reduce the problem to the 2-dimensional problem, which we can solve by our algorithm. As soon as we find a face that contains a crossing of lines, we can stop and we do not need to check all the faces. However, if we do not need to split the hyper-cube, we will have to check every single face of the hyper-cube.

The work we will have to do is basically determining all the values on all the edges of the hyper-cube by binary search. A hyper-cube of dimension m has $m \cdot 2^{m-1}$ edges.

What we have implemented so far is a simplified algorithm, that is just checking for the same corners. As it is impossible to go very deep into the recursion for time reasons, we need to stop the recursion at a certain depth and try to find the maximum by a heuristic search within the hyper-cube. As for the heuristic search within the hyper-cube, we could use the hyper-sphere maximization method. But so far, we have only implemented a simple random search.

Some general notes about this method:

4.3.3.1 Enumeration of the corner points of the hyper-cube

A d -dimensional hyper-cube has 2^d corner points, which we enumerate in the following way: Starting from the origin, we get the corner point ID's by going to any combination of possible steps.

going 1 step into x-direction: PointID + = $1 \cdot 2^0$

going 1 step into y-direction: PointID + = $1 \cdot 2^1$

...

going 1 step into d-direction: PointID + = $1 \cdot 2^{d-1}$

e.g. point (1,1,...1) has PointID = $1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + \dots + 1 \cdot 2^{d-1} = 2^d - 1$

4.3.3.2 Enumeration of the sub-cube corners

To create the sub-cubes, we are dividing every edge of the hyper-cube into two. Therefore we always have three corner points of the sub-cubes on any edge of the original hyper-cube. Note, the sub-cube point ID's are *not* identical with the corner ID's of the original hyper-cube.

Enumeration:

going 1 step into x-direction: PointID + = $1 \cdot 3^0$

going 1 step into y-direction: PointID + = $1 \cdot 3^1$

going 1 step into y-direction: PointID + = $1 \cdot 3^2$

...

going 1 step into d-direction: PointID + = $1 \cdot 3^{d-1}$

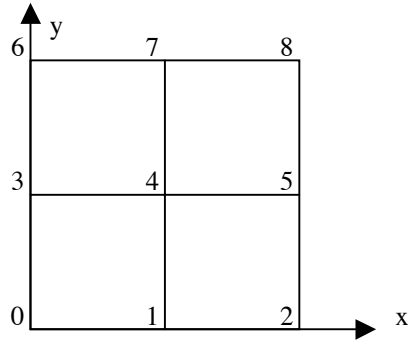


Figure 4.11: enumeration of the sub-cube corners

4.3.3.3 Construction of sub-hypercubes-ID's

To split a hyper-cube, we need to know the ID's of the sub-cubes. Later on, we will construct the sub-cubes by their ID's. Computing the ID's of the sub-cubes:

We always identify a sub-cube within a hyper-cube by its lowest sub-cube corner ID. For example, in 2-dimensions, there exist 4 sub-squares with ID's 0,1,3 and 4. We get the sub-cube ID's by every combination of going 1 step into any dimension. We can describe this process by a binary tree. The leaves of the tree are the ID's of the sub-cubes.

e.g. 3 dimensions:

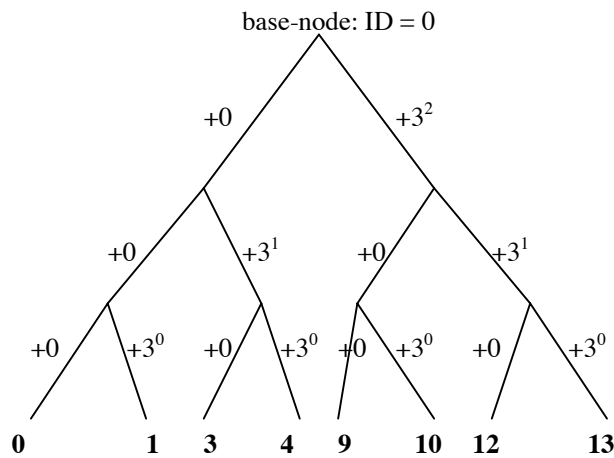


Figure 4.12: tree to generate the ID's of all the sub-cubes within a hyper-cube

From this tree, we can see that subcube[0] has ID 0; subcube[1] has ID 1; subcube[2] has ID 3; ... In 3 dimension, the sub-cubes have the ID's 0,1,3,4,9,10,12 and 13.

4.3.3.4 Construction of a sub-hypercube by its ID

From the above algorithm, we get the ID's of the sub-cubes. Now, we want to construct the sub-cubes by their ID's. Again, we can construct the sub-cubes by the construction of a binary tree: ID of the sub-cube is our base-point and from this base point we get the corresponding points by every combination of going 1 step into any direction:

example: construction of the 3-dimensional sub-hypercube with ID=12



Figure 4.12: tree to construct a sub-hypercube by its ID

From this binary tree, we get the corner points of the sub-hypercube with ID 12:
corner point[0] = 12; corner point[1]=13; ...

4.3.4 Simulated Annealing

Simulated Annealing is a minimization method from statistical physics. This minimization method is also popular for the computation of approximations of NP-complete problems. The idea behind *Simulated Annealing* is to slowly reduce the "energy" of a function. We just want to give a short summary of the algorithm:

Starting in a random point p_k , the function is going to be evaluated at $p_k' = p_k + v$, where v is a random vector. To determine whether p_k or p_k' is our new point, we need to compute the difference between their function values:

$$\Delta F = F(p_k') - F(p_k)$$

We choose p_k' as our new point with probability $p(\Delta F)$, and we keep our previous point p_k with probability $1-p(\Delta F)$.

where $p(x)$ is the following probability density:

$$p(x) = \begin{cases} 1, & \text{if } x \leq 0 \\ \exp(-x/T) & \text{otherwise} \end{cases}$$

T is the "temperature" that will slowly be decreased over time. The decrease of T can be chosen as follows:

$$T_k = \frac{A}{\log(1+k)}$$

where A is a constant value (initial "temperature"), depending on the problem.

From that follows that if we find a point with a smaller value, we are always changing our current point into the point with smaller and therefore better value. If the new point p_k' has a higher value, we are still going to that new (but worse) point with a certain probability. The reason for this is that if we are in a local minimum, we

might not be able to escape from that local minimum just by trying to improve our minimum value if the radius of the random vector is too small.

In this work, we are only using the *Simulated Annealing* method for comparison reasons, therefore we have not spent a lot of time to optimize the method-parameters.

4.3.5 Comparison the n -dimensional maximization methods

We also compared the n -dimensional maximization methods with the following settings:

Random directions	number of restarts	20
	number of re-tries when we are in a local maximum	5
Hyper-spheres method	number of restarts	100
	K	50
Hypercube¹ method	max. depth	4
	number of random points per hyper-cube	100
Simulated Annealing²	number of function evaluations	200'000
	A (initial "temperature")	100

Table 4.2: settings of the maximization methods

Method	avg. maximum found	Average number of evaluations
Random directions	20.048	315'204
Hyper-spheres method	20.184	78'149
Hyper-cube method	19.925	249'736
Simulated Annealing	19.794	200'000

Table 4.3: result of the comparison

The results from the single comparisons can be found in the appendix. The functions we maximized were score-functions of a prediction process with linear models (3 dim.)

The *Hyper-spheres* method turned out to be the best method with only about 1/3 of the function evaluations of all the other methods.

The worst method in this comparison turned out to be *Simulated Annealing*, but we can not say that the *Simulated Annealing* method is worse than the other methods as we have not spent a lot of time adapting the parameters of *Simulated Annealing* to the problem.

The main problem of the *Hyper-cube method* was that we just went down to recursion depth 4 and then did a simple random search.

The *Random directions method* was surprisingly good in this case, but with a huge number of function evaluations.

¹ simplified version with random search after depth 4

² not optimized with any parameters, such as initial temperature

5. MODELS & RESULTS

5.1 model 1: mean & standard deviation of past prices

In this model, we make an estimation for the highest and the lowest price of the present day. To estimate the highest price of the current day, we compute the mean and standard deviation of the price difference between opening prices and highest prices over the most recent 10 days. Accordingly we estimate the lowest price of the present day by the mean and standard deviation of the difference from the opening prices and lowest prices over the most recent 10 days. The basic idea is to always buy at the lowest price and to sell at the highest price of the day.

If we were allowed to make 2 transactions per day and we always succeeded to both buy and sell every day, we would not have to bother about any trend, as we would make a gain every day (under the assumption that we could also "sell in short", which means that we can sell stocks before we buy).

For this model, we are just using one single parameter p_0 , which we use as a factor for the standard deviation.

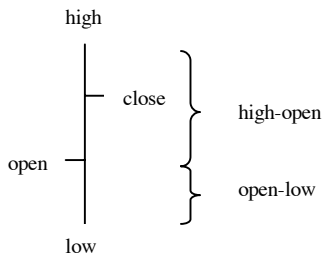


Figure 5.1: representation of the daily data

$$buyPrice_i = \min(open_i, open_i - mean_{open-low} - p_0 \cdot stddev_{open-low})$$

$$sellPrice_i = \max(open_i, open_i + mean_{high-open} + p_0 \cdot stddev_{high-open})$$

A simulation over 125 days with this strategy leads to the following result:

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
Adobe	16. Jun 99	10. Dec 99	74.75 %	2769.47	83.32	66.28	78.64	-6.65	56.78	55.48
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	1.70	3.74	24.46	3.96	11.94	28.36
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	-7.29	-4.02	7.16	-16.82	-5.44	-9.78

Table 5.1: simulation of model 1

At first glance, this model seems to be good for 25 training-days (5 weeks). But table 5.2 shows that in a simulation with other stocks we did not succeed with this strategy.

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]		
	Startdate	Enddate			20d	25d	30d
AT&T	25. Jun 99	21. Dec 99	-0.75 %	700.16	-5.60	-16.30	2.73
Boeing Air	23. Jun 99	17. Dec 99	-9.56 %	499.05	0.02	-14.57	-6.38
Microsoft	16. Jun 99	10. Dec 99	18.74 %	732.44	3.84	-0.69	15.69
Hewlett Packard	25. Jun 99	21. Dec 99	19.55 %	1692.63	-9.55	-23.29	0.07
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	-5.27	-6.77	-15.16
Coca-Cola	25. Jun 99	21. Dec 99	-6.93 %	519.41	-18.64	-19.02	-24.77

Table 5.2: second simulation of model 1

Analysing the good results from model 1:

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
Adobe	74.75%	34.55%	24.72	5	83.31%	23	8	2.04%	3.68	62	61
				25	78.64%	18	5	2.66%	4.83	61	62
General Motors	14.89%	7.99%	11.50	25	24.46%	17	3	1.27%	2.93	58	65
				125	28.36%	14	5	1.61%	2.72	55	69
Disney	-3.58%	-0.66%	12.40	25	7.15%	16	13	0.29%	2.85	56	65
AT&T	-0.75%	0.72%	14.14	30	2.73%	7	7	0.28%	4.31	56	65

Table 5.3: validation tests of model 1

5.2 model 2: computing the trend by a linear least squares fit

In this model, we compute the current trend by a linear least squares fit on the closing prices of the most recent 4 days. We are just using 4 days and therefore 4 data points to fit a straight line, as otherwise the change in our computed trend might be too slow and we would not be able to react to changes at the right time.

From the discrete linear least squares fit, we get the best fitted line $y = ax + b$ for the most recent 4 closing prices. The gradient a of this line is equivalent to the trend.

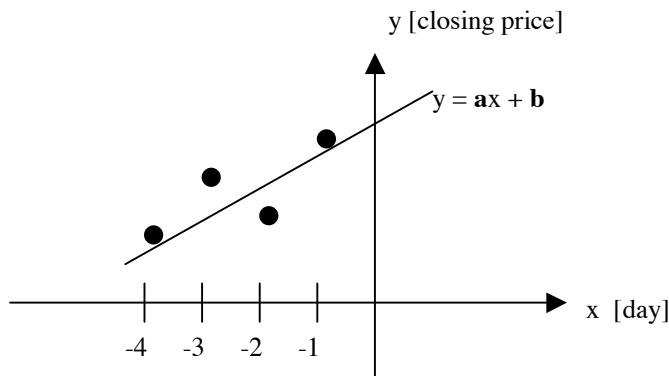


Figure 5.2: computation of the trend

To make a decision, we use one parameter p_0 :

if (trend $\geq p_0$) then BUY else SELL

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	10.21	9.73	10.63	-8.76	8.84	1.41
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	-4.44	11.43	-4.43	-15.09	3.70	-7.72
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	13.14	5.50	0.49	-8.45	3.84	-9.95

Table 5.4: simulation of model 2

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	5	10.21%	9	12	0.53%	3.79	66	53
				10	9.73%	8	11	0.53%	3.10	67	52
				25	10.63%	7	7	0.81%	4.53	65	54
				75	8.48%	9	6	0.58%	2.77	66	53
Disney	-3.58%	-0.66%	12.40	5	13.14%	4	11	0.96%	5.47	64	57
				10	5.50%	7	9	0.43%	4.45	67	53
				75	3.84%	5	7	0.40%	4.34	65	55

Table 5.5: validation tests of model 2

From the high standard deviations of the transaction gains, we must assume that the good results were a coincident.

5.3 model 3: estimation of $high_i$ and low_i by extrapolation

The idea of this model is similar to the previous model. This time we make an estimation for the highest ($high^*$) and the lowest price (low^*) of the current day.

Both low^* and $high^*$ are computed by a discrete linear least squares fit on the closing prices of the most recent 4 days.

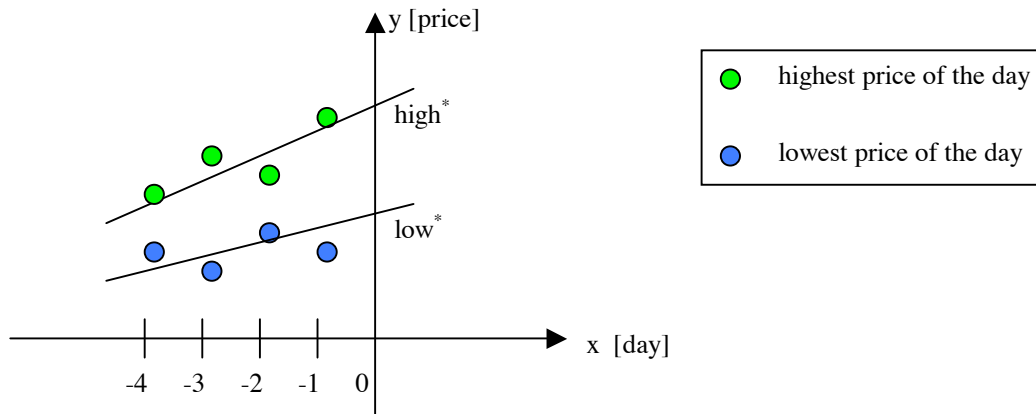


Figure 5.3: estimation of high and low at the current day

From the two discrete linear least squares fits ($y = ax + b$), we get b , which we can use to extrapolate $high^*$ and low^* :

$$high_i^* = b_{high}$$

$$low_i^* = b_{low}$$

$$sellPrice_i = \max(open_i, open_i + p_0 \cdot (high_i^* - open_i))$$

$$buyPrice_i = \min(open_i, open_i - p_1 \cdot (open_i - low_i^*))$$

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	-0.63	2.63	-3.84	7.06	9.30	9.39
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	-5.03	16.23	13.22	4.45	1.79	6.80
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	1.70	-1.17	5.44	-1.48	-1.01	9.98

Table 5.6: simulation of model 3

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	75	9.30%	26	21	0.21%	2.06	56	64
				125	9.39%	26	26	0.20%	2.23	61	61
General Motors	14.89%	7.99%	11.50	10	16.23%	19	15	0.49%	3.10	54	69
Disney	-3.58%	-0.66%	12.40	25	5.44%	23	16	0.17%	2.48	62	61
				125	9.98%	24	24	0.23%	2.40	59	64

Table 5.7: validation tests of model 3

5.4 model 4: optimal combination of different trends

In this model, we make decisions according to trends of different lengths. The length sizes are 3, 4, 6 and 10-days. The trend lengths are chosen rather short as we want to avoid very similar trend values from one day to another day. The trends are computed by a discrete linear least squares fit on the opening prices.

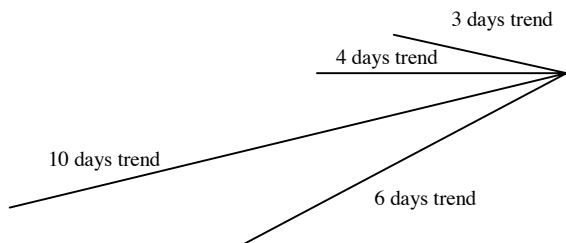


Figure 5.4: trends of different time periods

The gradient (which is equal to the trend) of the fitted lines could be any value and therefore have a weight which is too high compared to the other trends. Therefore we are transforming these trend-values into values $\in]-1,1[$ at first.

For this purpose, we use the Sigmoid function:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

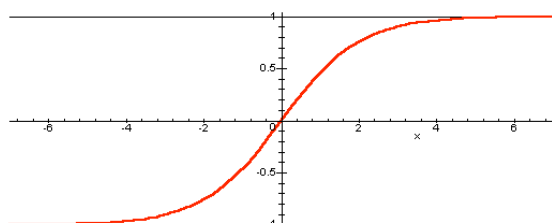


Figure 5.5: Sigmoid function

$decision = p_0 \cdot f(trend_3) + p_1 \cdot f(trend_4) + p_2 \cdot f(trend_6) + f(trend_{10})$
 if (decision > 0) then BUY else SELL

Stock	Simulation period		Buy/Hold upper limit		Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	21.35	6.27	5.60	14.87	19.52	9.40
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	12.16	23.02	-2.47	-0.49	21.60	18.24
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	-9.92	11.31	12.03	-1.34	-2.79	-10.80

Table 5.8: simulation of model 4

At first glance, this model seems to be pretty good, but analysing the good results shows that we have a very high standard deviation of the gains from the single transactions.

	buy&hold	random decisions (1000 simulations)		Training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	5	21.35%	12	6	1.13%	3.35	66	53
				75	19.52%	18	10	0.67%	2.43	66	53
General Motors	14.89%	7.99%	11.50	10	23.02%	20	11	0.70%	2.53	63	58
				75	21.60%	15	8	0.90%	3.13	66	55
				125	18.24%	12	6	0.97%	2.60	64	57
Disney	-3.58%	-0.66%	12.40	10	11.31%	15	12	0.43%	2.73	74	47
				25	12.03%	11	11	0.58%	3.75	69	52

Table 5.9: validation tests of model 4

5.5 model 5: p% filter Rule

In practice, there exists a strategy called "p% filter rule", where p is 4% for example. The strategy itself is quite simple: We buy as soon as the price climbs more than p% from the most recent minimum (over a certain time period). We sell as soon as the price falls more than p% from the most recent maximum (over the same time period).

With this strategy, we are always going with major trends. We are not trying to sell exactly at a local maximum and to buy exactly in a minimum.

However, this is not necessarily always a good strategy: Imagine, the price is climbing p%, then falling p% and climbing again for p%. In such a case, we always buy in a local maximum and sell in a local minimum.

Our approach is slightly different. In our model, we are trying to optimize p. Instead of using just one p, we are using two parameters: p_0 for buying and p_1 for selling:

$$\text{buyPrice}_i = \min(\text{open}_i, (1 + p_0) \cdot \min_{5 \text{ days}})$$

$$\text{sellPrice}_i = \max(\text{open}_i, (1 - p_1) \cdot \max_{5 \text{ days}})$$

where $p_0, p_1 \in] -\infty, +\infty [$

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	7.89	-3.50	-3.55	3.67	-5.99	-19.44
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	-5.68	7.09	0.46	-0.16	1.52	27.87
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	15.51	-0.53	8.00	3.63	-1.27	-11.53

Table 5.10: simulation of model 5

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
General Motors	14.89%	7.99%	11.50	125	27.86%	21	10	0.82%	2.21	68	55
Disney	-3.58%	-0.66%	12.40	5	15.51%	8	10	0.91%	4.75	66	55
				25	8.00	10	6	0.54%	3.64	58	64

Table 5.11: validation tests of model 5

5.6 model 6: transactions with gain - guarantee

In this model, we are not going to sell before we made a certain gain, even though we might have to wait for a long time or in very bad cases even forever. After selling, we are waiting until the price is falling. We are not going to buy again before the price fell $p_1\%$. It would not make sense to buy at a higher price than we have sold before, otherwise *buy & hold* would have been the better choice.

Applying this strategy, we are sure to make a gain with every complete transaction (buy action and sell action).

Of course, there also exist problems: We might buy at a very high price. After we bought, the price is falling and never reaches the same high price-level again. In that case, we are loosing our money by keeping the stocks and waiting for better times.

There is another problem: Once we sold, we are waiting until the price is falling below our previous selling price, but this might never happen. This is not a severe problem, as we just miss the chance to make gains, but we are not loosing money.

We can express the buyprice and the sellprice at *day i* as follows:

$$\text{BuyPrice}_i = (1 - p_0) \cdot \text{latestSellPrice}$$

$$\text{SellPrice}_i = (1 + p_1) \cdot \text{latestBuyPrice}$$

with $p_0, p_1 \geq 0$

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	13.33	8.30	15.42	15.29	3.18	17.03
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	8.25	26.40	7.25	6.80	2.29	23.50
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	4.25	27.31	-3.58	-3.58	9.33	19.05

Table 5.12: simulation of model 6

Looking at the transaction sequence of Disney with 25/50 training-days, we can see that we bought at the beginning and were expecting the stock-price to climb, but it never rose. So we were not able to commit any transactions except for selling at the end of the simulation, which ended in the same result as *buy & hold*.

In general, we can say that the more transactions we manage to make, the better the result will be. The good thing about this strategy is that we can not be worse than the *buy & hold* strategy if the performance of the *buy & hold* strategy itself is negative. Looking at table 5.13, we can see that we were always making good transactions, except for the very last transaction (selling at 0).

day	date	action	price
-124	25. Jun 99	buy at	39.9375
-121	30. Jun 99	sell at	41.0625
-120	01. Jul 99	buy at	40.8125
-115	09. Jul 99	sell at	42.875
-104	26. Jul 99	buy at	41.4375
-87	18. Aug 99	sell at	41.5
-85	20. Aug 99	buy at	40.875
-83	24. Aug 99	sell at	41.25
-82	25. Aug 99	buy at	41
-81	26. Aug 99	sell at	41.9375
-78	31. Aug 99	buy at	41.125
-73	08. Sep 99	sell at	41.6875
-46	15. Okt 99	buy at	41.1875
-45	18. Okt 99	sell at	42.125
-41	22. Okt 99	buy at	41.75
-33	03. Nov 99	sell at	43.0625
-7	10. Dec 99	buy at	43.0625
0	21. Dec 99	sell at	41.625

Table 5.13: transaction sequence of McDonalds

In table 5.13 we can see that the model was working very well as we could make a lot of transactions. Unfortunately, in most of the cases, we will end up with just a few transactions; and at the end with our last transaction, we will lose all our relatively small gain from the few transactions.

In general, we can say that this model will be working well for stocks that do not climb or fall too fast. It will work excellent for stocks that have a rather high volatility, but whose trend is increasing or decreasing just slowly.

A possible solution for the problems with this model could be that we are selling after 20 days of no transaction. We hope that we can regain our committed loss by restarting our strategy on a deeper price level. We are also buying after 20 days of no transaction on a higher price level.

A simulation with this new strategy leads to the following result (simulation: 300 days)

Stock	Startdate	Enddate	Buy/Hold	upper limit	5d	10d	25d	50d	75d	125d
McDonalds	14. Okt 98	21. Dec 99	30.36 %	11591.7	6.89	63.66	5.03	16.31	9.61	16.34
General Motors	12. Okt 98	17. Dec 99	71.62 %	17787.8	49.49	14.86	45.88	91.76	37.64	10.83
Disney	14. Okt 98	21. Dec 99	14.41 %	24318.6	13.71	12.15	13.73	12.23	31.23	11.77

Table 5.14: simulation of model 6 with a slightly different strategy (buy and sell after 20 days of no action)

Another try, where we only make restarts after 20 days of no buy-action (only good transactions possible!):

Stock	Startdate	Enddate	Buy/Hold	upper limit	5d	10d	25d	50d	75d	125d
McDonalds	14. Okt 98	21. Dec 99	30.36 %	11591.7	22.19	63.66	5.56	19.18	29.18	17.89
General Motors	12. Okt 98	17. Dec 99	71.62 %	17787.8	51.81	19.16	26.74	71.13	44.22	16.38
Disney	14. Okt 98	21. Dec 99	14.41 %	24318.6	4.45	-6.28	41.97	5.04	2.92	-8.17

Table 5.15: simulation of model 6, where we only buy after 20 days of no action

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	5	13.33%	6	0	2.12%	1.63	64	57
				10	8.30%	4	0	2.01%	0.54	63	56
				25	15.42%	8	1	1.63%	2.34	64	56
				50	15.29%	7	0	2.06%	1.40	64	55
				125	17.03%	8	0	1.99%	1.02	62	57
General Motors	14.89%	7.99%	11.50	10	26.40%	8	0	2.98%	1.54	68	54
				125	23.50%	6	1	3.64%	3.94	74	47
Disney	-3.58%	-0.66%	12.40	10	27.31%	11	1	2.25%	2.78	72	49
				125	19.05%	8	0	2.22%	1.89	69	52

Table 5.16: validation tests of model 6 (original strategy)

As we are enforcing the model to make always gains, it is not surprising that the single transaction gains have a relative high mean and a low standard deviation. But the main problems of this model are the two problems discussed above. Note, with the two new, slightly different strategies, we lose the guarantee that we cannot be worse than the *buy & hold* strategy if it leads to a negative performance.

5.7 model 7: combination of indicators from *Technical Analysis*

In this model, we are combining 3 popular overbought/oversold indicators from *Technical Analysis*. We are trying to find the optimal combination of three single decisions. At first, all three indicator values are transformed into a decision value $\in]-1,1[$ by the Sigmoid function. The advantage of this model is that we are not relying on just one single indicator. The model will always adapt to the currently best indicator, under the assumption that any of these three indicators is good (which is not necessarily the case).

$$decision1 = -f\left(\frac{RSI - 50}{10}\right)$$

$$decision2 = -f\left(\frac{PRoC - 100}{10}\right)$$

For decision 3, we use the change in the *On-Balance Volume* indicator. To get the rate of change, we make a discrete linear least squares fit on the indicator value. (∇ onBalanceVol)

$$decision3 = f(\nabla\text{onBalanceVol})$$

Using 3 parameters p_0, p_1 and p_2 , we are combining these 3 single decisions into one decision:

$$decision = p_0 \cdot decision1 + p_1 \cdot decision2 + p_2 \cdot decision3$$

if (decision ≥ 0) then BUY else SELL

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	3.14	23.12	3.92	9.75	9.73	9.49
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	11.81	12.03	9.90	19.16	22.43	-4.96
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	-2.75	29.16	5.25	-9.23	11.80	-1.09

Table 5.17: simulation of model 7

At first glance, this model seems to be pretty good, but a further simulation with other stocks for 8, 10, 15, 70, 75, 80 days shows this good result was probably just coincidence.

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			8d	10d	15d	70d	75d	80d
AT&T	25. Jun 99	21. Dec 99	-0.75 %	700.16	15.83	2.29	11.21	2.41	1.56	10.54
Adobe	16. Jun 99	10. Dec 99	74.75 %	2769.47	36.70	19.00	54.81	14.63	21.01	32.47
Coca-Cola	25. Jun 99	21. Dec 99	-6.93 %	519.41	-26.09	-20.50	-16.46	-19.91	-20.37	-14.86
Microsoft	16. Jun 99	10. Dec 99	18.74 %	732.44	9.55	23.12	24.53	4.85	11.77	18.37
Boeing Air	23. Jun 99	17. Dec 99	-9.56 %	499.05	-0.68	-9.25	2.42	-3.27	0.74	-4.96
Hewlett Packard	25. Jun 99	21. Dec 99	19.55 %	1692.63	-20.51	9.89	-3.84	2.39	-12.42	-11.94

Table 5.18: another simulation of model 7 with new stocks

Analysing the good values:

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	10	23.12%	14	10	0.91%	3.04	67	52
General Motors	14.89%	7.99%	11.50	50	19.16%	6	6	1.57%	4.63	63	58
				75	22.43%	7	4	1.92%	3.70	63	58
AT&T	-0.75%	0.72%	14.14	8	15.83%	12	10	0.75%	4.06	65	54
				15	11.21%	10	11	0.59%	4.28	62	57
				80	10.54%	6	4	1.37%	9.32	55	64
Microsoft	18.74%	10.13%	13.06	10	23.12%	14	8	1.00%	3.26	69	56
				15	24.53%	10	8	1.30%	3.96	65	60
Disney	-3.58%	-0.66%	12.40	10	29.16%	10	10	1.35%	3.60	74	47
				75	11.80%	10	7	0.78%	5.26	66	55

Table 5.19: validation tests of model 7

5.8 model 8: Indicators from *Technical Analysis* (second model)

This model is also based on three different indicators from *Technical Analysis*. Contrary to model 7, we get only either 1 (BUY) or -1 (SELL), but no numeric values in between.

We are using the following 3 decisions:

decision 1: MACD indicator **BUY**, if the *fast MACD-line* crosses the *slow MACD-line* from below
SELL, if the *fast MACD-line* crosses the *slow MACD-line* from above
DO_NOTHING otherwise

decision 2: Stochastic-indicator **BUY**, if the fast *Stochastic indicator line* (*%K-line*) crosses the slow indicator line (*%D-line*) from below
SELL, if the fast *Stochastic indicator line* (*%K-line*) crosses the slow indicator line (*%D-line*) from above.
DO_NOTHING otherwise

decision 3: Exponential moving average **BUY**, if the price crosses its exponential moving average (0.95%) from above
SELL, if the price crosses its exponential moving average from below
DO_NOTHING otherwise

trend We use the trend, which is computed by a linear least squares fit on the closing prices of the most recent 5 days, as a constant.

$$decision = p_0 \cdot decision1 + p_1 \cdot decision2 + p_2 \cdot decision3 + f(trend)$$

where $f(x)$ is the Sigmoid function

if ($decision \geq 0$) then **BUY** else **SELL**

Stock	Simulation period		Buy/Hold	Upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	-6.52	-18.01	-2.58	-9.49	-21.67	-10.48
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	10.84	17.67	-0.01	9.11	7.55	15.06
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	33.03	47.29	31.19	2.77	-0.64	-0.02





Table 5.20: simulation of model 8

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
General Motors	14.89%	7.99%	11.50	10	17.67%	14	9	0.73%	2.18	62	59
Disney	-3.58%	-0.66%	12.40	5	33.03%	10	10	1.53%	4.67	68	53
				10	47.29%	12	9	1.95%	4.38	73	48
				25	31.19%	8	8	1.83%	5.15	70	51

Table 5.21: validation tests of model 8

5.9 model 9: relationship volume / trend

The idea behind this model is that the trading volume should confirm the trend. If both trend and volume are rising, it is a good sign. If the trend is falling and the volume rising, it is a sign to sell as quickly as possible. If the trend is rising, but the volume is falling, it is a sign that the trend might not be as strong as it seems to be. We can distinguish between the following four cases:

	trend dprice/dt 	trend dprice/dt 
volume dvol/dt 	+++	---
volume dvol/dt 	+	-

We compute both trend ($dprice/dt$) and the change in the volume ($dvol/dt$) by a linear least squares fit on the past N days, where we choose N as parameter:

$$N = \max(2, \text{floor}(p_0))$$

We can transform the above table into a decision value $\in]-1,1[$ by the Sigmoid function:

$$\text{value} = f\left(p_1 \cdot \left[1 + \text{sign}\left(\frac{dvol}{dt}\right) \cdot f\left(\frac{dvol}{dt}\right)\right] \cdot \frac{dprice}{dt} + p_2\right)$$

if (value ≥ 0) then BUY else SELL

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	28.71	0.50	21.11	4.19	1.70	-9.57
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	9.28	-6.60	20.01	9.73	8.65	-0.74
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	18.78	-6.13	16.11	11.57	14.33	15.53

Table 5.22: simulation of model 9

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]		
	Startdate	Enddate			20d	25d	30d
AT&T	25. Jun 99	21. Dec 99	-0.75 %	700.16	-11.13	3.77	5.27
Boeing Air	23. Jun 99	17. Dec 99	-9.56 %	499.05	-15.91	-24.88	-18.51
Microsoft	16. Jun 99	10. Dec 99	18.74 %	732.44	21.10	-3.28	-9.81
Hewlett Packard	25. Jun 99	21. Dec 99	19.55 %	1692.63	-14.33	-13.76	6.40
Adobe	16. Jun 99	10. Dec 99	74.75 %	2769.47	33.85	25.74	4.40
Coca-Cola	25. Jun 99	21. Dec 99	-6.93 %	519.41	-17.53	-6.99	-3.14

Table 5.23: simulation of model 9 with other stocks

further validation tests:

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	5	28.71%	14	12	1.00%	2.47	65	54
				25	21.11%	10	6	1.28%	3.90	66	53
General Motors	14.89%	7.99%	11.50	25	20.01%	9	7	1.19%	3.10	62	59
Disney	-3.58%	-0.66%	12.40	5	18.78%	12	14	0.71%	3.05	75	46
				25	16.11%	11	10	0.70%	3.96	68	53
				50	11.57%	11	10	0.58%	2.59	64	57
				75	14.33%	10	7	0.86%	2.97	62	59
				125	15.53%	6	3	1.70%	4.33	69	52

Table 5.24: validation tests of model 9

5.10 model 10: decision - model

There is no special idea behind this model.

$$openV = \sum_{i=0}^5 0.8^i \left(\frac{open_{today-i}}{open_{today-1-i}} - 1 \right)$$

$$highLowV = \sum_{i=1}^5 0.8^{i-1} \left(\frac{high_{today-i}}{low_{today-i}} - 1 \right)$$

$$closeV = \sum_{i=1}^5 0.8^{i-1} \left(\frac{close_{today-i}}{close_{today-1-i}} - 1 \right)$$

$$decisionValue = p_0 \cdot openV + p_1 \cdot closeV + p_2 \cdot highLowV + (open_i / open_{i-5} - 1)$$

if (decisionValue ≥ 0) then BUY else SELL

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30	600.96	-6.45	1.69	-3.09	9.26	14.57	6.89
General Motors	23. Jun 99	17. Dec 99	14.89	627.01	11.64	4.57	1.12	-10.97	-16.87	9.73
Disney	25. Jun 99	21. Dec 99	-3.58	736.64	13.93	12.96	5.12	3.14	-8.96	-7.13

Table 5.25: simulation of model 10

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	50	9.26%	15	9	0.39%	2.28	63	56
				75	14.57%	15	9	0.58%	1.83	66	53
Disney	-3.58%	-0.66%	12.40	5	13.93%	10	16	0.60%	4.57	73	48
				10	12.96%	11	9	0.65%	2.97	69	52

Table 5.26: validation tests of model 10

Amazing in this model is Disney with 5 training days. Although there were 73 good decisions against 48 bad decisions (60% correct decisions), we made only 10 good transactions against 16 bad transactions.

5.11 model 11: best trading days

In this model, we are only trading on certain days within a week, e.g. buying on Monday, selling on Friday. From the training, we can find out which were the best trading days to buy and to sell within the last month for example. If it turns out that we should have always bought on Mondays and sold on Fridays, we are also going to trade on these days this week. But maybe it also turns out that we should not have bought or sold at all.

The maximization will be very fast as we only have a few plateaus in the score-function (36, for each of the two parameters, there are 5 different days, +1 for DO_NOTHING).

We have two (discrete) parameters in this model, one for the buying-day, one for the selling-day:

Parameter 1: buying day

Parameter 2: selling day

if (day mod 5) == buyingDay then BUY

if (day mod 5) == sellingDay then SELL

else DO_NOTHING

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
Adobe	16. Jun 99	10. Dec 99	74.75	2769.47	52.39	34.63	55.94	61.68	35.18	60.93
General Motors	23. Jun 99	17. Dec 99	14.89	627.01	4.25	17.08	3.84	6.39	-9.49	8.77
Disney	25. Jun 99	21. Dec 99	-3.58	736.64	0.05	-8.14	6.84	-1.18	7.81	6.94

Table 5.27: simulation of model 11

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
General Motors	14.89%	7.99%	11.50	10	17.08%	7	5	1.40%	4.16	57	64
Disney	-3.58%	-0.66%	12.40	25	6.84%	8	6	0.54%	3.85	66	55
				75	7.81%	9	11	0.42%	3.18	61	60
				125	6.94%	8	13	0.39%	3.91	65	56

Table 5.28: validation tests of model 11

5.12 model 12: one-price model

This model does not have any deeper background. For each day, we get a price, which is a buyprice or a sellprice according to our current state. For this model, we are using three parameters: p_0, p_1 and p_2 .

$$\text{actionprice} = \frac{p_0 \cdot \text{open}_i \cdot (\text{open}_{i-1} - p_1 \cdot \text{close}_{i-1})}{\text{high}_{i-1} \cdot \text{low}_{i-1} - p_2 \cdot \text{high}_{i-1} \cdot \text{close}_{i-1}}$$

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30%	600.96	5.84	-4.54	8.03	-10.70	-5.46	-12.57
General Motors	23. Jun 99	17. Dec 99	14.89%	627.014	-2.92	7.69	1.85	6.02	2.06	5.27
Disney	25. Jun 99	21. Dec 99	-3.58%	736.639	9.93	3.76	-8.30	-9.84	8.22	8.13

Table 5.28: simulation of model 12

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	25	8.03%	6	2	0.98%	1.58	61	58
Disney	-3.58%	-0.66%	12.40	5	9.93%	11	7	0.62%	4.44	61	60

Table 5.29: validation tests of model 12

The values for the transaction gains (mean and stddev) do not say very much about the model as the model made only a few transactions.

5.13 model 13: prediction of supply and demand

As stock prices are driven by the ratio between supply and demand, we try to put this idea into a model. This model itself is based on a differential equation, similar to the Volterra principle.

We describe the change in the number of bid and ask with the 4 unknown parameters p_0, p_1, p_2 and p_3 :

$$\begin{aligned} bid'(t) &= p_0 \cdot bid(t) + p_1 \cdot bid(t) \cdot ask(t) \\ ask'(t) &= p_2 \cdot ask(t) + p_3 \cdot bid(t) \cdot ask(t) \end{aligned}$$

In the training period these four parameters are adapted to yield the maximum gain.

Solving this equation for $bid(t)$ and $ask(t)$, we predict supply and demand of the current day. If $bid(t)$ is less than $ask(t)$, we are buying as prices normally will climb in such a case, on the other hand, if $bid(t)$ is higher than $ask(t)$, we are selling as prices are likely to fall.

To compute $bid(t)$ and $ask(t)$, we need to solve this differential equation numerically. To solve this equation, it is very important to choose a stable method as we need stable values for $bid(t)$ and $ask(t)$ up to 125 training days. A simulation with the Euler-Cauchy method (order 2) and stepsize $h = 1/400$ shows that already after only a few days, the curve becomes unstable. Contrary to that, with a Runge-Kutta method of order 4 we get a stable curve upto 125 days, even with a smaller stepsize (1/200):

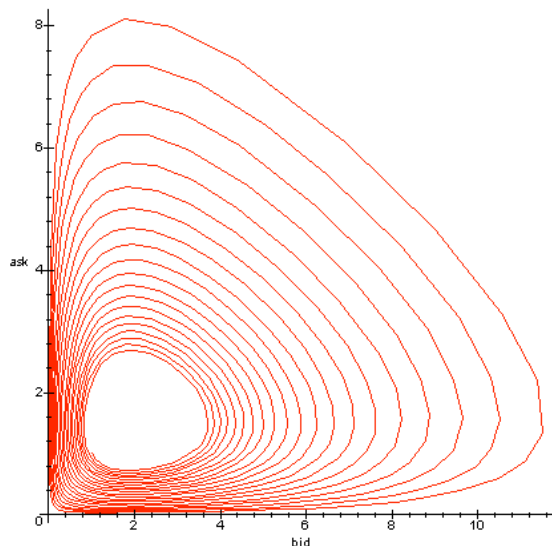


Figure 5.6: differential equation solved by Euler Cauchy (order 2), 20 days with stepsize 1/400

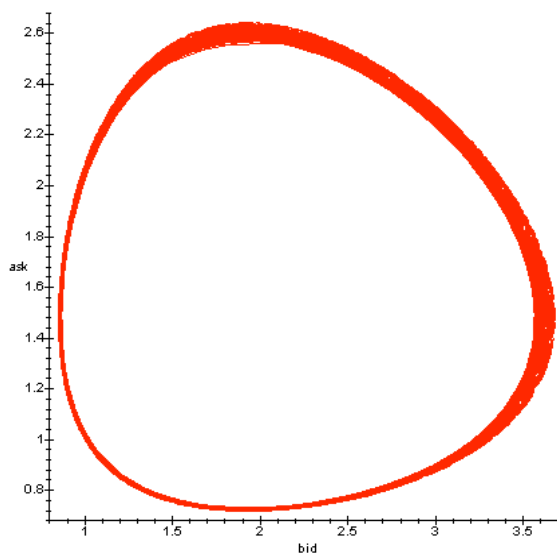


Figure 5.7: Exactly the same differential equation, but this time solved by a Runge-Kutta method of order 4. 125 days with stepsize 1/200

This model turned out to be excellent in the training.

This model is computationally very expensive. The computation of a simulation with 50 training days took more than 1 week.

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30%	600.96	31.36	0.33	19.07	0.80	-2.00	xxx
General Motors	23. Jun 99	17. Dec 99	14.89%	627.014	4.48	30.62	21.39	-6.79	xxx	xxx
Disney	25. Jun 99	21. Dec 99	-3.58%	736.639	20.57	16.39	-0.58	-2.13	xxx	xxx

Table 5.30: simulation of model 13

This model seems to be quite good for a rather short training period. Some further simulations with other stocks gave the following result:

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]				
	Startdate	Enddate			5d	8d	10d	25d	50d
At&T	25. Jun 99	21. Dec 99	-0.75%	700.16	-6.91	-9.05	4.04	19.63	17.90
Boeing	23. Jun 99	17. Dec 99	-9.56%	499.05	-15.63	-18.81	-19.42	-12.73	xxx
Adobe	16. Jun 99	10. Dec 99	74.75%	2769.47	10.74	31.43	49.39	59.82	34.70

Table 5.31: simulation of model 13 with new stocks

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	4.30%	2.55%	10.67	5	31.36%	16	7	1.25%	3.38	71	48
				25	19.07%	18	12	0.62%	2.85	66	53
General Motors	14.89%	7.99%	11.50	10	30.62%	18	10	0.99%	2.60	71	50
				25	21.39%	18	11	0.75%	2.30	68	53
Disney	-3.58%	-0.66%	12.40	5	20.57%	16	10	0.81%	4.35	75	46
				10	16.39%	12	16	0.63%	4.22	70	51
AT & T	-0.75%	0.72%	14.14	25	19.63%	19	11	0.64%	3.05	71	48
				50	17.90%	13	12	0.76%	4.60	61	58

Table 5.32: validation tests of model 13

5.14 model 14: a random based model

In this model, we leave the responsibility of how to make a decision to the model. If the model finds out that in the training, the best way to make a decision was a random decision, we are also going to make a random decision. But if the model finds out that we still should make random decisions, but we that should buy more often than we should sell, we are also going to make a random decision, but with a random generator that is choosing BUY more often than SELL.

$$decision = uniform(p_0) - p_0/2 + p_1$$

if (decision \geq 0) then BUY else SELL

where $p_0 \in [0,1]$ and $p_1 \in [-1,1]$.

The smaller the value of p_0 , the less random the model becomes. If p_1 is equal to zero, then the decisions are completely random, if $p_1 \geq p_0/2$, our decision will always be BUY and if $p_1 < -p_0/2$ our decision will always be SELL.

So the model itself can determine the amount of randomness.

In this model, we have the problem of evaluating the score-function with exactly the same arguments, the score will be different, as the decisions are random. Therefore we will never get the same transaction sequence with the same parameters.

For that reason we need a special maximization method. We are not just evaluating the function at one single point, but at many points around a single point, hoping that there exist areas in the score-function with a high score-average and areas with low score-averages. We determine the function value at a certain point as the average of many function values close to that point. From that reason, the maximization process is computationally very expensive.

Stock	Simulation period		Buy/Hold	upper limit	Training period					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30	600.96	-8.12	-6.28	-7.97	-14.97	-4.36	-15.97
General Motors	23. Jun 99	17. Dec 99	14.894	627.014	-10.90	4.04	-12.05	4.93	1.96	13.91
Disney	25. Jun 99	21. Dec 99	-3.58176	736.639	1.18	-6.84	-14.66	-9.70	-0.82	-8.56

Table 5.33: simulation of model 14

This model shows that our considerations about random models in chapter 2.4 were right: Models that are not significantly better than random models will not be able to outperform the *buy & hold* strategy in general.

5.16 model 16: pattern matching

The following model is different from the other models; we do not have any parameters. To make a decision, we always use all the past data (in our case data from the past 10 years).

In this model, we are looking for the 5 most similar situations of the past to the current situation. We define a situation as the sequence of gains and losses (in %) from the past week. The days within a situation are exponentially weighted, which means that the most recent days have a higher weight than the gains/losses a week ago. Finally, from the 5 most similar situations of the past 10 years, we make a decision, which is weighted according to the similarity (distance) to the present situation.

The decision value is computed according to the 5 nearest neighbours of the present situation:

$$decisionValue = \sum_{i=1}^5 weight(i) \cdot decision(situation(i))$$

The weights of the nearest neighbours are computed as follows. The more similar the situation is (the smaller the distance), the higher the weight for this decision:

$$weight(i) = \frac{\frac{1}{dist(situation(i), present)}}{\sum_{k=1}^5 \frac{1}{dist(situation(k), present)}} = \frac{1}{\sum_{k=1}^5 dist(situation(k), present)} \cdot \frac{1}{dist(situation(i), present)}$$

To compute the distance between two situations, we only consider the daily gains in percent. The distance from a situation in the past to the present situation is then computed as follows (the importance of the days are exponentially decreasing (with factor 0.8)):

$$dist(situationA, situationB) = \sum_{i=0}^N 0.8^i \cdot comparison(lastday(situationA) - i, lastday(situationB) - i)$$

where $N = \#days$ in a situation, e.g. 5 days, or we also could choose $N \rightarrow \infty$, as the weights are exponentially decreasing

The weights of the comparisons of the days within a situation are exponentially decreasing. This makes sense as we want that the days closer to the end have a higher weight in the fitting process.

The comparison of two days are equal if the earnings have the same value. Note, we are not computing the earning of a certain day according to the opening and closing price, as the opening price of the following day is not necessarily the same as the closing price of the previous day, and we could gain or loose money over night.

We are comparing two days according to the following policy: The more similar the days are, the less the distance will be:

$$comparison(dayA, dayB) = \left(\frac{open(dayA)}{open(dayA-1)} - \frac{open(dayB)}{open(dayB-1)} \right)^2$$

Unfortunately, the simulation with the pattern matching strategy seems to be rather bad. One reason is that, in general, we were not able to find very similar situations in the past. We might have to use more data than only 10 years (?).

Stock	Simulation period		Buy/Hold	upper limit	Result on Test-set, values in [%]
	Startdate	Enddate			
McDonalds	25. Jun 99	21. Dec 99	4.30	600.96	-3.04
General Motors	23. Jun 99	17. Dec 99	14.90	627.01	4.48
Hewlett Packkard	25. Jun 99	21. Dec 99	19.55	1692.63	19.92
Coca Cola	25. Jun 99	21. Dec 99	-6.93	519.41	8.85
At&T	25. Jun 99	21. Dec 99	-0.75	700.16	-0.51
Microsoft	16. Jun 99	10. Dec 99	18.74	732.44	8.26
Apple	16. Jun 99	10. Dec 99	122.10	4301.45	33.90
Adobe	16. Jun 99	10. Dec 99	74.75	2769.47	-5.29
Boeing	23. Jun 99	17. Dec 99	-9.56	499.05	3.05
Disney	25. Jun 99	21. Dec 99	-3.58	736.64	-22.22

Table 5.34: simulation of model 16

In spite of the rather disappointing result, we are expecting somehow a good ratio between good and bad decisions. But this ratio is rather disappointing too:

Stock	good decisions done	bad decisions done
McDonalds	59	60
General Motors	57	64
Hewlett Packkard	67	57
Coca Cola	67	54
At&T	57	62
Microsoft	65	60
Apple	62	62
Adobe	53	70
Boeing	62	58
Disney	52	69

Table 5.35: ratio between good and bad decisions

5.17 model 17: Majority decision of independent models

In this model, we use 5 independent models to make a decision. Every single model will be optimized by itself with a different number of training days. The decision is always the majority decision of these 5 models.

However, majority decisions do not necessarily need to be better than single decisions:

Example [Bernasconi99]: 3 models, every model can make 60% correct predictions:

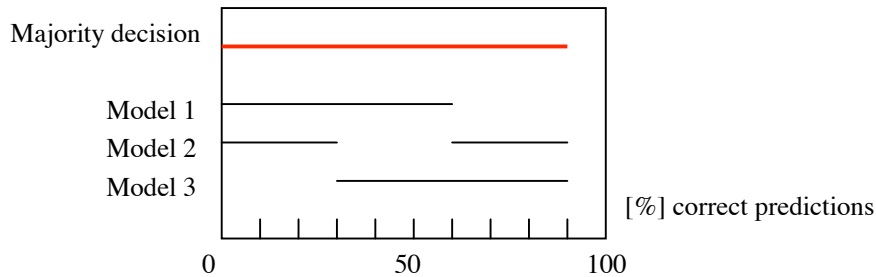


Figure 5.8: best case, majority decision is in **90%** of all the cases correct

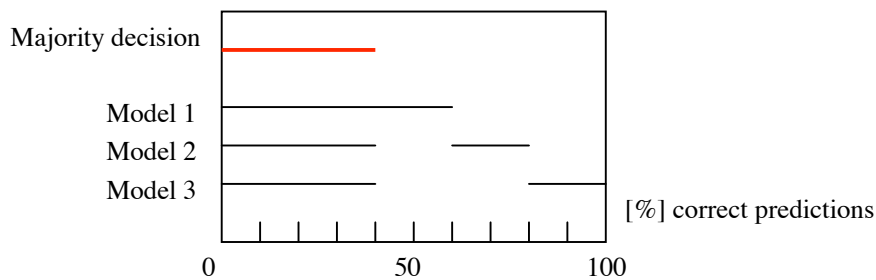


Figure 5.9: worst case, majority decision is only in **40%** of all the cases correct, although every single model is in 60% of all the cases correct.

We can generalise this example: with M models (M odd), which are all correct in P percent of all the cases, we can determine the best case and the worst case of a majority decision:

$$best = \min(100, M / \lceil M/2 \rceil \cdot P)$$

$$worst = \frac{P - 100 \cdot \lfloor M/2 \rfloor / M}{1 - \lfloor M/2 \rfloor / M}$$

First of all, we can see that a majority decision can only be good, if the models are in more than 50% of all the cases correct. From the example above, we can also see, that it is rather bad, if the models are very similar: Assume we have two models that are making exactly the same decision in any case. In that case, a third model can not do anything against these two models, even though it is right.

From that reason, we are choosing models with completely different ideas:

Model	number of training days	correct decisions in the simulated models		number of parameters
Model 4	13 days	10d training: 54.0%	25d: 51.8%	(3)
Model 7	12 days	10d training: 56.0%	25d: 52.1%	(3)
Model 8	15 days	10d training: 51.8%	25d: 50.6%	(3)
Model 9	25 days	10d training: 50.4%	25d: 54.3%	(3)
Model 18	8 days	5d training: 53.2%	25d: 53.2%	(2)

Table 5.36: correct decisions

We need to simulate this model with new stocks, as we have simulated the single models with our standard stocks (DIS,MCD,GM) already. It would not be correct to use the same data again, as we know the results from the single models with this data already.

Stock	Simulation period		Buy/Hold	upper limit	Result, values in [%]
	Startdate	Enddate			
Hewlett Packkard	25. Jun 99	21. Dec 99	19.55	1692.63	15.31%
At&T	25. Jun 99	21. Dec 99	-0.75	700.16	11.37%
Microsoft	16. Jun 99	10. Dec 99	18.74	732.44	16.67%

Table 5.37: result from the simulation

5.18 model 18: supply and demand as functions of cosine and sine

In this model, we predict supply and demand by a function of sine and cosine. For this model, we assume that there exists a kind of dynamic cycle in the price, which can be approximated by a sine and cosine function. This time, we describe the number of bid(t) and ask(t) as follows (with 2 parameters):

$$bid(t) = p_0 \cdot \sin(p_1 \cdot t)^2 \cdot \cos(t)$$

$$ask(t) = \sin(p_1 \cdot t) \cdot \cos(t)^2$$

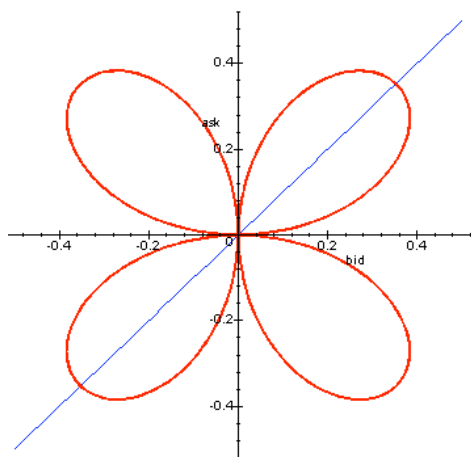


Figure 5.10: $bid(t)$ and $ask(t)$ with $p_0=1.0$ and $p_1=1.0$

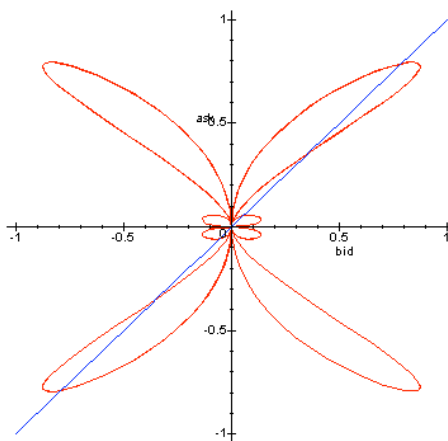


Figure 5.11: $bid(t)$ and $ask(t)$ with $p_0=1.0$ and $p_1=3.0$

Walking along the curve, the decision swaps whenever the curve crosses the $y=x$ line. With p_0 , we can stretch the curve such that the decision limits are changing.

Contrary to model 13, we do not need to make several time steps for every day and we do not have any troubles with stability, as we know the exact function. We do not need to keep any state from the training.

As this model was relatively easy to compute, we were simulating this model for 300 days:

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	14. Oct 98	21. Dec 99	30.35 %	1010.06	4.26	38.31	44.19	26.07	20.35	7.37
General Motors	12. Oct 98	17. Dec 99	71.62 %	1452.00	19.32	36.31	34.90	58.72	-0.30	-5.56
Disney	14. Oct 98	21. Dec 99	14.41 %	1516.69	50.89	-15.84	3.92	18.89	48.58	33.00

Table 5.38: simulation of model 18

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	30.35%	16.02%	19.59	10	38.31%	43	27	0.50%	2.58	151	140
				25	44.19%	37	26	0.63%	3.07	150	141
Disney	14.41%	9.53%	23.89	5	50.89%	37	35	0.67%	4.58	157	129
				75	48.58%	37	37	0.59%	3.46	151	134
				125	33.00%	38	40	0.41%	3.18	156	130

Table 5.39: validation tests of model 18

In the simulation of Disney, 5 training days, the model was able to make 55% correct predictions (up/down). But in general, the results are rather disappointing as the training itself was always excellent. Even though we are just using 2 parameters, we were able to reach the maximum gain (for decision models) in 204 out of 300 times with 10 training days! A short summary of the training (log-files from Disney):

Disney:

number of training days	number of maximum gain reached (in the training)	average ratio of reach gain vs. maximum possible gain
5	299 out of 300 times	99.94 %
10	204 out of 300 times	97.95 %
25	0 out of 300 times	79.06 %
50	0 out of 300 times	60.38 %
75	0 out of 300 times	47.75 %
125	0 out of 300 times	31.47 %

Table 5.40: number of maximum possible gain reached in the training

Encouraged from the good training result, we are repeating the simulation with other stocks and this time with only short training periods:

(note, the upper limit is chosen as upper limit for decision models)

Stock	Simulation period		Buy/Hold	upper limit	Training period, values in [%]					
	Startdate	Enddate			4d	6d	8d	12d	15d	20d
Microsoft	16. Jun 99	10. Dec 99	18.74 %	201.77	12.61	8.63	12.56	3.94	17.88	3.88
Coca-Cola	25. Jun 99	21. Dec 99	-6.93 %	167.46	1.24	6.20	18.26	-4.62	-0.75	-9.44
AT & T	25. Jun 99	21. Dec 99	-0.75 %	201.82	13.35	-14.33	4.75	27.59	16.37	21.87

Table 5.41: simulation of model 13 with other stocks

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
Coca-Cola	-6.93%	-2.54%	12.23	8	18.26%	14	15	0.64%	3.66	69	52
AT & T	-0.75%	0.72%	14.14	12	27.59%	16	11	1.06%	5.92	65	54
				15	16.37%	20	15	0.50%	3.67	64	55
				20	21.87%	20	14	0.65%	3.80	61	58

Table 5.42: validation tests of model 13

Considering the training results, the final result of this model were quite disappointing. Maybe this model shows that we can not make any conclusions from the training to the real world.

5.19 model 19: extension of model 1

This model is an extension of model 1. In model 1, we were not bothering about any trends. In this model, we are using a short term trend as a kind of reinforcement of the price.

In short, if we have an upward price trend, we are willing to buy at a rather high price and we do not want to sell unless it is a really high price. On the other hand, if we have a downward trend, we are willing to sell at a rather deep price and we do not want to buy unless it is a really deep price.

For the reinforcement, we are simply using the difference between the number of upward price changes and downward price changes from the most recent 5 days as estimation of the current trend.

up: number of upward price changes of the most recent 5 days

down: number of downward price changes of the most recent 5 days

$$\text{buyPrice}_i = \min(\text{open}_i, \text{open}_i - 0.5 \cdot \text{mean}_{\text{Open-Low}} + p_0 \cdot (\text{up-down}) \cdot \text{stddev}_{\text{Open-Low}})$$

$$\text{sellPrice}_i = \max(\text{open}_i, \text{open}_i + 0.5 \cdot \text{mean}_{\text{High-Open}} + p_1 \cdot (\text{up-down}) \cdot \text{stddev}_{\text{High-Open}})$$

Stock	Simulation period		Buy/Hold	upper limit	Training period , values in [%]					
	Startdate	Enddate			5d	10d	25d	50d	75d	125d
McDonalds	25. Jun 99	21. Dec 99	4.30 %	600.96	-0.99	4.49	15.94	-5.72	-18.53	1.09
General Motors	23. Jun 99	17. Dec 99	14.89 %	627.01	-4.30	-5.88	0.83	5.85	14.48	8.69
Disney	25. Jun 99	21. Dec 99	-3.58 %	736.64	4.34	5.25	26.36	16.54	19.18	13.43

Table 5.43: simulation of model 19

Analysing the good results:

	buy&hold	random decisions (1000 simulations)		training days	gain with our model	transactions		transaction gains		decisions	
		mean	stddev			good	bad	mean	stddev	good	bad
McDonalds	14.89%	7.99%	11.50	25	15.94%	19	19	0.41%	2.23	57	64
Disney	-3.58%	-0.66%	12.40	25	26.36%	24	16	0.63%	2.97	66	58
				50	16.54%	23	17	0.43%	3.12	55	68
				75	19.18%	23	16	0.50%	3.30	60	65

Table 5.44: validation tests of model 19

5.20 model 20: statistical approach

To explain the idea behind this model, let us have a look at figure 5.12:

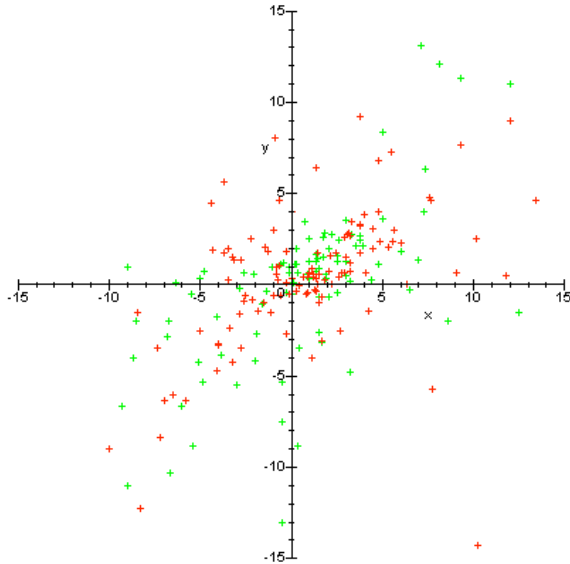


Figure 5.12: red: buy-points, mean $(2.817/1.311)$; green: sell-points, mean $(1.652,0.605)$

At first glance, all the points in figure 5.12 look like randomly distributed. But computing the mean and standard deviation of the red and the green points, we will find out that the mean of the red points and the mean of the green points is different, even though we are taking 2000 points.

In this model, we are computing the mean and standard deviation of the green points (sell-points) and the red points (buy-points). For every training day, we have one point, which is either a buy-point or a sell-point, depending on the best decision at that day. To make a decision at the present day, we compute the point in the xy -plane for the present day. If the distance from the point of the present day is closer to the mean of the buy-points, our decision is BUY, otherwise SELL.

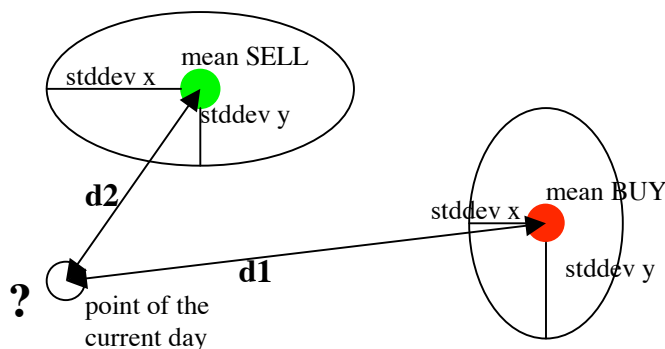


Figure 5.13: SELL point: mean of all the sell-points in the xy plane; BUY point: mean of all the buy-points in the xy -plane

Whenever we compute any distance to the BUY-point or SELL-point, we are normalizing the distance in x -direction by dividing through the standard deviation in x and accordingly in y -direction.

If the BUY-point and the SELL-point are very close to each other, our predictions will not be very good, as in that case all the points are more or less random. In general, we can say, the closer the mean of the buy-points and the mean of the sell-points, the more random our predictions will be. Therefore, we are maximizing the distance

from the SELL point to the BUY point in the training. To maximize the distance between these two points, we let the model choose the data:

$$x = \frac{\text{open}(\text{day}) - \text{open}(\text{day} - D1)}{\text{open}(\text{day}) - \text{close}(\text{day} - 1)}$$

$$y = \frac{\text{open}(\text{day}) - \text{open}(\text{day} - D2)}{\text{open}(\text{day}) - \text{close}(\text{day} - 1)}$$

We are ignoring cases, where $\text{open}(\text{day}) - \text{close}(\text{day} - 1) = 0$. (does not happen very often)

In the training, the model chooses $D1$ and $D2$ such that the distance from the mean of the buy-points and the mean of the sell-points is maximized. In the simulation, we can observe that the model was often choosing $D1=1$ and $D2=2$, which makes sense.

However, the results are not that good:

Days: (=number of points in the xy-plane)	Disney	McDonalds	General Motors	Microsoft	Hewlett Packard
Buy & hold	-3.58%	4.30%	14.89%	18.74%	19.55%
15d	6.26% (62/57)	-9.64% (57/61)	6.45% (57/62)	10.58% (65/58)	8.98% (58/64)
20d	7.96% (66/53)	0.70% (59/59)	15.83% (63/56)	9.70% (57/66)	-8.58% (54/68)
50d	6.26% (62/57)	-1.71% (62/56)	11.29% (64/55)	-13.19% (52/71)	-4.01% (59/63)
75d	-0.43% (48/71)	-10.73% (52/66)	-3.97% (57/62)	7.28% (58/65)	-33.79% (46/76)
100d	0.56% (50/69)	2.69% (56/62)	-0.69% (58/61)	-9.55% (50/73)	-12.94% (55/67)
150d	9.32% (59/60)	-5.75% (55/63)	9.61% (63/56)	-1.96% (51/72)	-1.31% (56/66)
200d	11.90% (63/56)	9.11% (58/60)	-0.62% (60/59)	14.63% (63/60)	-4.09% (54/68)
500d	8.27% (61/58)	-5.25% (57/61)	12.3% (63/56)	6.17% (54/69)	13.13 (60/62)
1000d	13.53% (62/57)	-5.38% (57/61)	0.80% (57/62)	2.81% (56/67)	5.02% (57/65)
2000d	14.90% (65/54)	4.81% (59/59)	-3.98% (56/63)	0.60% (54/69)	-9.38% (52/70)

Table 5.45: results of model 20

The rather disappointing result can be explained by a further analysis of the points in the xy-plane: Even though we are maximizing the distance from the mean of the buy-points to the mean of the sell-points, this distance is not significantly higher than the distance between two random sets of points.

The main idea of this model was to find the two days of the past that have the most significant influence to the present day. This idea can be extended to any dimension, for example trying to find the 10 most significant days of the past to make a prediction for the current day. But then, there will be the problem of over-fitting. How many days should we use and which information of the past should we choose? Then we would have to optimize the ratio of the number of variables and the maximum increase of the distance these variables can cause: we would have to find as less variables as possible that increase the distance most.

Another idea could be to have two means for both buy-points and sell-points: We distinguish between "normal" buy-points and "strong" buy-points and accordingly for sell-points.

6. IMPLEMENTATION

6.1 Program design

The main goal of the design was to have a general minimization/maximization framework on the one hand. On the other hand, our program should be extendible with new models. The design for the models can be described in a simplified version as follows:

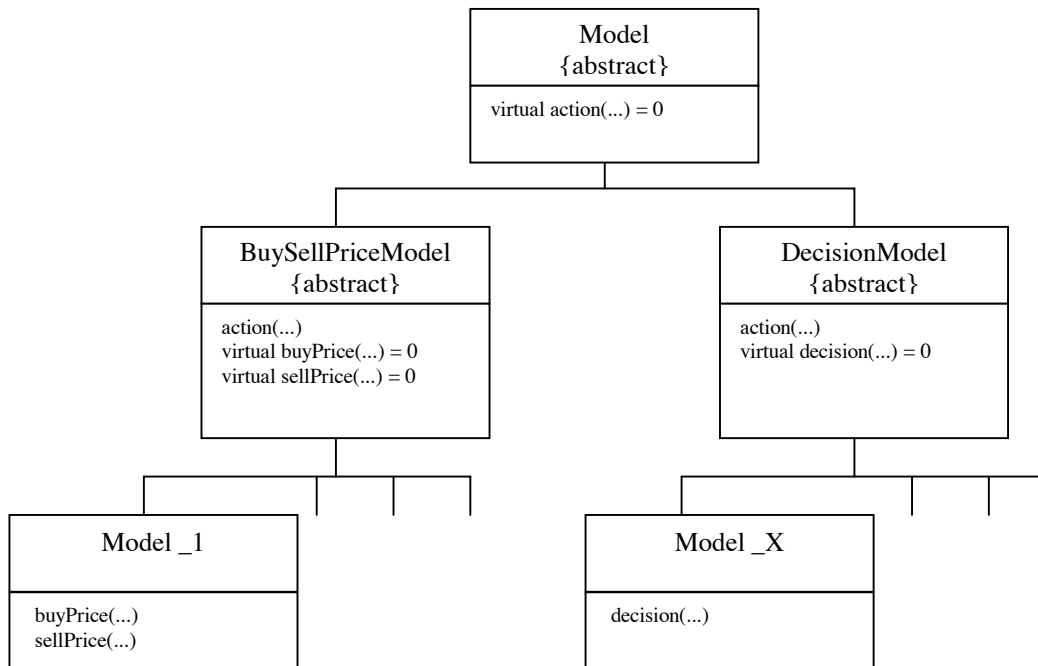


Figure 6.1: model design

All that needs to be encoded in the different models is the formula or program to compute a buyprice and sellprice in the case of buy/sellPrice models, and in the case of decision models a decision according to the desired formula. All the work to compute the new states and whether we can buy or sell at our desired price is done by the abstract classes.

The design of functions and the optimization of functions is done in a very general way and could easily be extended for the maximization and minimization for any kind of function. In order to define a function, we would just like to call a maximization and minimization method without bothering about the correct optimization strategy. Of course, if we want to have a special maximization strategy, we can choose to do so. For that reason, we are using a strategy manager, where all the different maximization strategies are registered and which always chooses the right strategy for the right dimension. The design of a function with its maximization strategy can be described as follows:

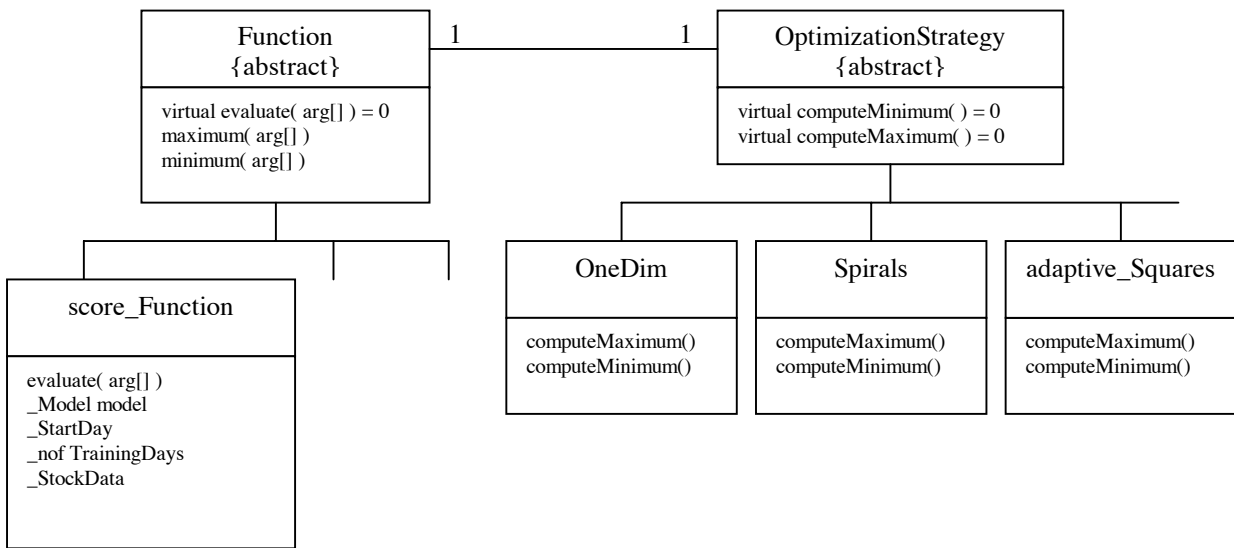


Figure 6.2: function and maximization design

The decision which maximization strategy to choose for a certain function is done by the StrategyManager:

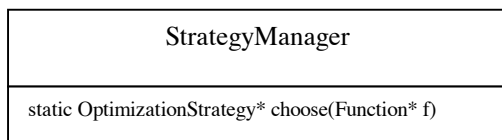


Figure 6.3: strategy manager

The StrategyManager is a class where all the different optimization strategies are registered. The static method `choose(Function*f)` chooses the registered optimization strategy for the appropriate dimension of the function.

Now we can easily maximize the score-function: at first we need to define the score function object:

```

TrainingSetFunction scorefunc(signed startday, signed nofTrainingdays,
                              const Model& model, const StockData& stock);
  
```

to get the optimal score and the optimal parameters, we can call

```

double max = scorefunc.maximum(arg)
  
```

where *arg* is a list of the optimal parameters returned. We do not have to bother about the right amount of parameters, everything is done by the abstract classes.

6.2 Extension with new models

To define a new model, the abstract class *DecisionModel* or *BuySellPriceModel* has to be extended. Models are considered to be stateless, but if necessary, a state can be defined, which has to be given to the model. A range for the parameters should be given, as a hint for the maximization process.

```

class MyModel: public DecisionModel() {
public:
    MyModel();
    DECISION decision(double arg[], signed day, const StockData, ModelState** modelState);
    double getArgRangeMin(double argMin[]);
    double getArgRangeMax(double argMax[]);
}
  
```

To implement a model:

```
class MyModel::MyModel(): DecisionModel(3) {}
```

This means that MyModel is a decision model containing 3 parameters. Now we need to implement the policy to make a decision:

```
DECISION MyModel::decision(double arg[], signed day, const StockData& stock, ModelState** state)
{
    if (stock.getOpen(day-1) > arg[0]*stock.getClose(day-1) ) {
        ....
        return SELL;
    }
    return BUY;
}
```

And finally, to simulate our model:

```
TestSetFunction testset( signed startDay, signed nofSimulationDays, signed nofTrainingDays,
                        const Model& model, const StockData& data);
```

calling

```
double gain = testset.gain()
```

we get the gain on the test-set with our model and the according stock data.

6.3 Extension with new stocks

The historical price data of stocks are stored in text-files available at *finance.yahoo.com*. If it is a .csv file, we need to store it as text file (no commas). The .csv file can be loaded in MsExcel for example and then stored as text-file.

There is a special class to load the stock data from a text-file called *class StockData*. In the constructor, the filename of the text-file containing the data has to be given. e.g.

```
const StockData& stock("data/daily/adbe.txt");
```

The class will load the data and compute all the different indicators in advance such that calling these indicators is not an expensive function call any more, but only a memory access.

6.4 Verification of the program

The maximization process can be verified by the use of two completely different maximization strategies for the same function. Keeping the maximization process running for a long time enough, we are supposed to get the global maximum. If there was a bug in the maximization process, we would not get the same maximum. To find out whether the arguments of the maximum value are correct, we can simply evaluate the function with these arguments and we are supposed to get the maximum value (e.g. with a program written in a different language such as Maple).

To verify the results we got from the simulations, we can use the log-files. For every single simulation with K training days, stocks and models, there exist two log-files: One for the maximization process "*paramK.txt*" (K is the number of training days) and one log-file for the transactions of the simulation "*transacK.txt*".

The maximization log-file contains all the results of all the maximization processes of every day of the simulation:

e.g. model 5, Boeing Air, 50 training days, /model5/ba/param50.txt

day	upper limit	max found	arg1	arg2
-124	209.232	44.935	0.0428224	0.0159144
-123	207.069	32.443	0.0518796	-0.00214277
-122	154.734	27.1374	0.0240923	-0.0121036
-121	165.084	29.3647	0.0241018	-0.0121049
-120	157.276	28.4366	0.0241739	-0.0124546
...

Table 6.1: maximization log-file

There is also a log-file for every transaction done in the simulation. We can write a program (e.g. in Maple) that computes the gain according to this log-file. If there was a bug in the program, we would not get exactly the same result as the simulation got. A further advantage of this log-file is that we can also use it for validation tests.

e.g. model 5, Boeing Air, 50 training days, /model5/ba/transac50.txt

day	date	action	price
-124	23. Jun 99	buy	42.3125
-119	30. Jun 99	sell	42.5625
-116	06. Jul 99	buy	42.5625
-114	08. Jul 99	sell	43.75
-112	12. Jul 99	buy	43.8125
-109	15. Jul 99	sell	47.4375
-105	21. Jul 99	buy	44.625
...

Table 6.2: transaction log-file

As we have both log-files, we can also verify the prices: We have the optimal parameters from the maximization log-file and we have the formula of the model. We can evaluate the model with these parameters and are supposed to get the buy/ or sell price of the according day in the transaction log-file.

7. CONCLUSIONS AND FUTURE WORK

The main goal of this work was the development of maximization methods for piecewise constant functions. For functions in 1 dimension, there was known algorithm that finds the global maximum $O(P)$, with P plateaus. With the *Adaptive Squares* method, we have developed an efficient maximization algorithm that is also able to find the global maximum. We proposed an extension of this algorithm to any dimension. We have also developed an heuristic algorithm with spirals in two dimensions and with hyper-spheres for n dimensions that turned out to be quite good and fast.

We have seen that even very simple models can make a huge gain in the past. But in general, such models are not necessarily useful to make predictions for the future. Even though we were sometimes able to completely predict the stock market for 10 and more days with a model that contained only 2 parameters, we were not successful by applying this model to future data.

Sometimes we can read from people who found relations between the stock market and football games. This will be exactly the same problem: Although you can find relations in the past, you will mostly fail making predictions for the future.

We have showed that random models have mean gain of about half that of the *buy & hold* strategy. As the standard deviation of the mean gain is very high, it is rather difficult to show that a given, presumably not random model is not just good on the test-set by coincidence. We have also seen that random based models (models that make random decision according to a certain probability density) can not be good if they are not significantly better than real random models.

There are different explanations possible why we did not succeed to find a reliable model that is able to predict the market in general: First of all, we still do not know whether short-term price movements in the stock market are predictable at all. And secondly, our simulated stocks had a very high volatility: there were often daily price changes of 10% and more, which makes a big difference between a single good or bad decision. Furthermore models might not be good enough as they are all still rather simple models.

Comparing training periods, we can say that, in general, we got better results with rather short training periods (less than 50 training days in general).

A kind of surprise is that price models were not necessarily better than decision models, although price models can theoretically reach a much higher maximum gain than decision models. Comparing the score-functions of price models and decision models could give an explanation:

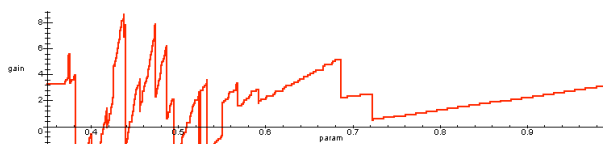


Figure 7.1: score-function of a typical buyprice/sellprice model

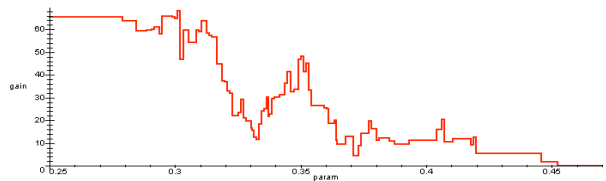


Figure 7.2: score-function of a typical decision model

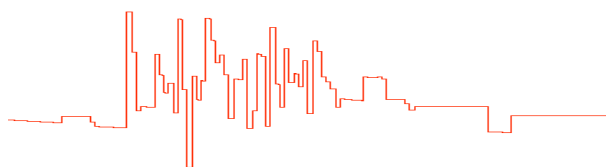


Figure 7.3: score-function of a one-price model

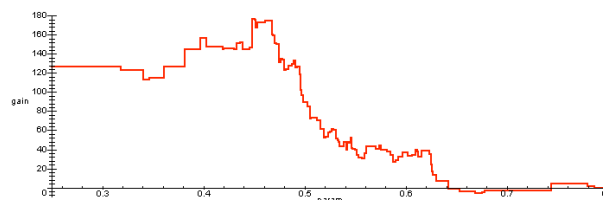


Figure 7.4: score-function of another decision model

Looking at the figures 7.1 - 7.4, we can see that score-functions of decision models are much "smoother", because from one plateau to another, there is only one decision changing in general. In price models (buyprice/sellprice models), any change (from buy to sell or the other way round) at a certain day can change the whole following sequence, because there are two completely different policies. In figure 7.3, we can see that only a

small change often caused a result from very good to very bad. In decision models, the curve is much smoother compared to price models and therefore, we can have some more trust that a small change in the parameter is still good in general.

Now, as we have a general prediction tool with good maximization methods, we can start developing more complex models, not only with technical data, but also with models containing fundamental data.

One approach in the future could be to predict price changes of a certain stock according to changing economic variables. Another approach could be to have a list of certain stocks, for example all the stocks of an industrial sector. Then we could train a model to pick the winner within this list of stocks according to changing economic variables.

As for the maximization methods, we need to prove that our considerations about the *Adaptive Hypercubes* method as extension of the *Adaptive Squares* method are correct. Furthermore, these algorithms could be extended to parallel algorithms.

8. REFERENCES

- [Schwarz93] H.R.Schwarz - *Numerische Mathematik*, 1993
- [Bernasconi99] Jakob Bernasconi - *neural networks* (lecture notes), 1999
- [Rice95] John A. Rice - *Mathematical statistics and data analysis*, second edition, 1995
- [Char91] Bruce W.Char, Keith O.Geddes, Gaston H. Gonnet, Benton L.Leong, Michael B. Monagan, Stephen M.Watt - *MapleV5 Library Reference Manual*, 1998
- [Press99] William H.Press, Saul A. Teukolsky, William T.Vetterling, Brian P.Flannery - *numerical recipes in C, the art of scientific computing*, second edition, 1999
- [Zweig97] Martin Zweig - *Winning on Wall Street* - 1997
- [Anderson97] Richard Anderson - *Market timing models*, (models from *Fundamental Analysis*), 1997
- [Thomsett99] Michael C. Thomsett - *Mastering Technical Analysis*, 1999
- [Plummer90] Tony Plummer - *Technical Analysis and the dynamics of price*, 1990
- [Murphy99] John J. Murphy - *Visuelle Aktienanalyse*, 1999

9. APPENDIX