ASTA
a Test Bench and Development Tool for Trading Algorithms

Thomas Hellström
Center of Mathematical Modeling (CMM)
Department of Mathematics and Physics
Mälardalen University
S-721 23 Västeras, Sweden
Technical Report Series IMa-TOM-1998-xx

August 27, 1998

Abstract

This report describes an implementation of **ASTA**, an Artificial Stock Trading Agent in the Matlab programming environment. The primary purpose of the project is to supply a stable and realistic test bench for development of multiple-stock trading algorithms. The behavior of the agent is controlled by a high level macro language, which is easily extendable with user defined functions. The buy and sell rules can be composed interactively and various types of data screening can be easily performed, all within the Matlab syntax. Various forms of performance evaluation and benchmarks are discussed.

Apart from being a test bench for trading algorithms, the system may be also run in batch mode where a supplied objective function maps a trading strategy to a profit measure. This can be used to tune parameters or to automate the development of trading strategies, for example with genetic methods. Some examples of tuning parameters in standard statistical indicators are presented.

To improve the performance of a given algorithm, the data from the simulated trades can be post processed by classification methods such as artificial neural networks or fuzzy rule bases.

The **ASTA** system has been applied successfully to historical stock data, and results covering 11 years of the Swedish stock market are presented.

Keywords: trading, artificial trader, technical analysis, data mining, prediction, stock returns.

[1] Also Department of Computing Science at Umeå University, Sweden. thomash@cs.umu.se

1 Introduction

The idea of expressing stock prediction algorithms in the form of trading rules has gained considerable attention in academic research in the last years. The international conference NNCM-96 devotes a whole section in the proceedings to "Decision Technologies". Bengio [2] writes about the importance to train artificial neural networks with a financial criterion rather than a prediction criterion. Moody and Wu [8] use reinforcement learning to train a trading system with objective functions such as profit, economic utility and Sharpe ratio. Atiya [1] describes a trading system based on time-variable stop-losses and profit objectives.

This report describes an implementation of ASTA, an Artificial Stock Trading Agent. Trading rule-based prediction algorithms may be easily evaluated here using historical data. ASTA also works as a development tool where trading rules can be combined and evaluated within the system.

The program has been developed in the Matlab programming language and can be used either as an ordinary objective function called from a user's program, or as an interactive tool for benchmarking and development of trading algorithms.

1.1 Objectives

The reasons for developing an artificial trader were:

1. A test bench for trading algorithms.

   Many "technical indicators" for stock prediction are accepted and widely used without having ever been subjected to an objective analysis with historical stock data. It is true that many commercial software packages for technical analysis offer both a comprehensive programming language and a simulation mode where the performance can be computed. However, most available products don't take this task very seriously and real trading simulation with a multiple stock portfolio is seldom possible. Furthermore, often the performance measures are not sufficient for a serious evaluation of an algorithm behavior over a longer period of time. Therefore, there is a need for an objective and scientific test bench for the methods and algorithms already developed and in common use.

   The need for proper evaluation of *new* trading algorithms is of course the same as for existing ones. ASTA is developed in Matlab 5 and is therefore mainly suitable for tests of algorithms developed in the same language.

2. An interactive development tool for trading rules.

   There are reason to believe that a successful trading system consists of many disjunct parts where a buy signal can be, for example, "screened" by looking at the traded volume. A buy signal issued with a low-traded volume may then be rejected. Other composite rules include looking at the general trend of the stock before accepting a signal from the system. ASTA provides the possibility to test such composite rules easily. The included function library and the possibility to define a trading strategy interactively makes implementation and evaluation of many trading strategies possible "without programming".

3. An non interactive development tool for trading rules.

   Furthermore, it is possible and possibly fruitful to automate the development of trading rules. Since ASTA defines the trading strategy as symbolic **Buy rules** and **Sell rules** given as arguments to the system, it would be perfectly possible to construct buy and sell rules, for example in a genetic framework.

   Even if the general look of the algorithm is fixed, there are often a lot of tunable parameters that affect the trading performance. Examples are filter coefficients, order of polynomials and levels above or below which an entity should pass in order to generate a trading signal. Since we believe that the actual behavior during a realistic trading situation is essential for proper selection and optimization of an algorithm, there is a need for an objective function that can be included in an optimization phase for parameter tuning.

4. Data generation for post processing

   The comprehensive and user-friendly macro language in ASTA makes it a very suitable tool for extracting data for further analysis, such as classification with neural networks or fuzzy rule bases. A raw selection of trading situations is first set up with simple trading rules. Data ("features) for these situations are then automatically written to a file. A "target" value for each trade is also supplied. Thereafter it will be a classification task to find out how to distinguish between a profitable trade and a non-profitable one, based on the given features.

2 Basic approaches to Stock predictions

Prediction algorithms for stock prices can be categorized in a number of ways. One categorization focuses on the way the points to predict are selected. Two broad classes can be identified; "The Time Series Approach" and "The Trading Rule Approach"

2.1 The Time Series Approach

The traditional way to define a stock prediction problem is to view the stock returns as a time series $y(t)$. One-day stock returns are often defined as

$$y(t) = \frac{Close(t) - Close(t-1)}{Close(t-1)} \tag{1}$$

To predict future return values, $y(t+h)$ is assumed to be a function g of the previous values. I.e.:

$$y(t+h) = g(y(t),\ y(t-1), ..., y(t-k)). \tag{2}$$

The task for the learning or modeling process is to find the function g that best approximates a given set of measured data.

The bias for the unknown function g can be chosen in many ways. Common choices are:

- A linear AR model:

$$g(t) = \sum_{m=0}^{d} a_m y(t-k) + a_{-1} \tag{3}$$

- A general nonlinear model implemented by a multi-layer-feed-forward neural network. A 1-hidden-layer net defines g as

$$g(t) = \sum_{j=0} \left(w_j \; h \left(\sum_{m=0} w_{m,j} y(t-k) + w_{-1,j} \right) + w_{-1} \right) \tag{4}$$

where h is a nonlinear function such as the sigmoid function $h(x) = \frac{1}{1+e^{-x}}$.

The unknown parameters (a_m for the AR model and $w_j, w_{m,j}$ for the neural network) are normally computed by the learning algorithm so that the root mean square prediction error

$$RMSE = \sqrt{\frac{1}{N} \sum_{t=1}^{N} (g(t) - y(t+h))^2} \tag{5}$$

is minimized.

It is most common to let the minimized RMSE measure (5) be the end point in the prediction task. However, in order to utilize the predictions, a decision-taking rule has to be created. It's common to formulate a simple rule like the following:

$$D(t) = \left\{ \begin{array}{l} \text{Buy : if } \; g(t) > \alpha \\ \text{Sell : if } \; g(t) < -\beta \\ \text{Do nothing : otherwise} \end{array} \right\} \tag{6}$$

where α and β are threshold parameters for buy and sell actions depending on the predicted change in the stock price.

The time series formulation based on the minimized RMSE measure (5) is not always ideal for useful predictions of financial time series for the following reasons:

1. The fixed prediction horizon h does not reflect the way in which financial predictions are being used. A model's ability to predict should not be evaluated at one single fixed point in the future. A big increase in a stock value 14 days into the future is as good as the same increase 15 days into the future!

2. The characteristics of the underlying processes that generate the time series are dramatically different in different parts in the series. Therefore, it is not a very good idea to attempt to fit a global model, however ingenious, which is fit for the entire time span. On the other hand, using a sliding window and computing models such as ARMA within the window is too local and does not make use of any knowledge from data outside the fixed window.

3. The RMSE measure (5) treats all predictions, small and large, as equal. This is not always appropriate. Predictions that would never be used for actual trading (i.e. price changes too small to be interesting) may cause higher residuals at the actual points of interest, in order to minimize the global RMSE.

4. A small predicted change in price, followed by a large real change in the same direction, will be penalized by the RMSE measure. This does not agree with the way we judge the quality of a stock prediction in actual trading, since the profit made by trading according to the prediction will just be higher if the actual change in price becomes larger than predicted.

5. Several papers report a poor correlation between the RMSE measure and the profit made by applying a prediction algorithm (e.g. Leitch & Tanner [5] and Bengio [2]). It has also been proven [7] that a strategy that separates the modeling from the decision-taking rule, such as the one in 6, is less optimal than modelling the decision-taking directly. Argument 3 and 4 both give explanations to these results.

2.2 The Trading Rule Approach

Instead of separating the prediction task and the decision task as was done in the "Time Series Approach", algorithms can be constructed to recognize situations where one should buy and sell stocks respectively. The task can be defined as a classification problem with three classes: "Buy", "Sell" and "Do nothing". Since we want classifications for all points 1 to N, we still need to express the predictions as time series. A trading rule can be described as a time series $T(t)$ defined as

attempting to model the time series and predict stock returns for all points 1 to N

$$T(t) = \left\{ \begin{array}{l} \text{Buy : if } g(\mathbf{X}(t)) > 0 \\ \text{Sell : if } g(\mathbf{X}(t)) < 0 \\ \text{Do nothing : otherwise} \end{array} \right\}. \tag{7}$$

The unspecified function g determines the type of the trading rule.

The argument $\mathbf{X}(t)$ has the following form:

$$\mathbf{X}(t) = (R_1(t), ..., R_N(t)) \tag{8}$$

where each $R_k(t)$ is an observable feature at time t. In the case of stock predictions it may be for example the k-day returns defined as

$$R_k(t) = 100 \cdot \frac{Close(t) - Close(t-k)}{Close(t-k)} \tag{9}$$

or standard technical indicators such as RSI, MACD etc.

This formulation covers the so-called "technical indicators", which are very common in real trading. Even if they are seldom generated in this formal way, they can often be formulated as a trading rule $T(t)$ as described above. One example is a moving average trading rule, that can be built from two moving average functions operating on the stock prices. Buy and sell signals are generated when two moving averages of different orders intersect.

The task for the learning process in The Trading Rule Approach is to find the function g so the profit is "maximized" when applying the rule on real data. Various ways to measure

the profit is discussed in section 3. Note the difference between this and The Time Series Approach, where the learning task was to find a function g so the $RMSE$ error (5) was minimized over the entire time series.

The Trading Rule Approach avoids many of the problems previously described with the Time Series Approach but does indeed have problems of its own:

1. Statistical significance. The trading rule $T(t)$ normally gives Buy or Sell signal only for a minor part of the points in the time series. While being one of the big advantages, it also presents serious statistical problems when computing levels of significance for the produced performance. It's no problem to find a rule $T(t)$ that historically outperforms any stock index as long as it doesn't have to produce more than a few signals. It's clear that a rule $T(t)$ that produces many signals for a given time interval, is more reliable than one that just produces a few, even if the profit is the same. However, it's *not* clear how this trade-off between profit and number of signals should be settled in an optimal way. The selection bias when choosing "good" trading rules makes a complexity argument worthless since even a very simple trading rule can produce very high profit if it just has to signal a few times. The general problem remains unsolved in the present work, with the single but important observation that the number of produced signals should be as high as possible in order to render the trading rule credible.

2. Commonly used methods for modeling or inductive learning, such as Regression models and Artificial Neural Networks cannot be applied in a normal fashion since a set of training data with Patterns and Targets are not available in the same way as in the Time Series Approach. However, a related modeling problem can be formulated as an extension to a system based on trading rules. The points where $T(t)$ signal *buy* or *sell* can be used to extract examples that can be input to a modeling or classification routine. The trading rule T may be refined in this way to produce a higher hit rate and most probably a higher overall profit. Therefore, a data extraction function has been built into the **ASTA** system.

The ASTA system is constructed according to the "Trading Rule Approach". In the next section the evaluation of a prediction system based on these principles will be investigated.

3 What Is Good Performance

A way to measure the performance of an algorithm is needed in two phases of the trading system's development cycle. Firstly, during the modelling phase where optimal settings on unknown parameters in the algorithm has to be decided. Secondly when the complete algorithm is put to test on historical data to see if it serves the original purposes of the development project. As was mentioned in section 2.1, there are good reasons to use the same performance measure in these two phases. The final result may otherwise be sub optimal.

Evaluation of prediction performance is an important, difficult, and often overlooked stage in the development of prediction algorithms for financial data. In the case of an artificial trader based on the "Trading Rule Approach" described in the previous section, one

problem is that the produced trades are comparatively few, which gives a somewhat weak statistical basis for our performance measures. Apart from this we have the problem with overtraining. By tuning the parameters in the artificial trader to maximize the performance on historical data, we always run a risk of fitting the algorithm too closely to the data set. Even if the trader's behavior is a rather "small" model (i.e. it has relatively few degrees of freedom), the problem must not be overlooked. Illustrative examples can be found in [6].

A third problem is the fact that the process is non-stationary, i.e. the performance of a trading behavior varies over time due to varying global conditions affecting the stock market as a whole. One strategy may work fine in a trending market, while another works best in a non-trending market.

Before starting to optimize the behavior or our artificial trader we must decide three things:

- What performance measure is relevant? Several options exist.
- What is the best way to measure the selected performance measure?
- With what do we compare our artificial trader's performance when we say that it is good? We need a good benchmark.

We start by investigating alternative performance measures:

3.1 Performance measures

3.1.1 Total Profit.

An intuitively appealing and very common measure is the total wealth achieved by the trader when simulating trading over the available training data period. The wealth is often presented as a function of time in a so called *equity diagram*. For comparison, a global stock index is often presented in the same diagram. The curves are scaled so the leftmost point has a wealth of 1 for both the trader and the index. The values for other points along the date axis can be then interpreted as wealth relative to the one at the starting point. A value 2.10 will mean, for example, that the initial capital has increased to 210% of the start capital. Therefore, the final value on the very right of the diagram will be the net result after completed trading for the entire time period. One example of an equity curve can be seen on the top diagram in figure 1. The major drawback of this method is that the trades in the beginning of the time period affect the end result more than the trades in the end of the time period. This is a consequence of the cumulative nature of the simulation. The profits in the beginning of the time period are being reinvested and therefore will appear "several times" in the total wealth resulting from trading during the entire time period. This is a nice effect if you want to show off your trading and stun people with an exponential increase in wealth. Its commonly used when the banks present results for their mutual funds over the years. However, the index curve is often not included in these cases since they would reveal that the mutual fund actually hasn't performed any better than the market in general. One has to realize that a constant economical growth of say 5% over a period of 10 years gives rise to a seemingly impressive exponential wealth curve for the average stock. With or without clever trading systems.

3.1.2 Average Profit Per Day

To get around the cumulative effect in the total-profit measure, one could compute the average profit per day (percentage) and use that as a measure of the trading performance. By this we lose the possibility to evaluate the trader's behavior during the varying global market conditions that arise during any longer simulation period.

3.1.3 Profit Per Year.

A compromise between using the total profit and the average profit per day would be offered by computing the annual profit achieved by applying the trading algorithm. The mean annual profit could be used as total measure for the entire time period. However, it's also very important to pay attention to the performance in each individual year.

3.1.4 Statistical Significance

As discussed in section 2.2, the profit from a Trading Rule based system has a very weak statistical significance if the number of trades produced during the simulation is too low. The number of trades are therefore of central importance and should be presented along with the trading results.

3.2 What is the best way to measure the selected performance measure?

Its important to realize that the profit measure we select for optimization and evaluation is a stochastic variable which has not got any single "value" that can be measured. Instead it has a probability distribution which can be described by its statistical moments such as the *expectation value* and the *variance*. What we get when we compute e.g. the annual profit for a year, is merely *one* sample from this distribution (assuming it is stationary over the years). The task of "maximizing" the profit by changing the parameters in the model now becomes less well defined. Obviously, the common way of using the *mean* value is just *one* option. Other choices could be the lower limit of a confidence interval, the *median value* or the *Sharpe Ratio.* The issue is further discussed in section 8 where the profit is examined as a function of various parameters in the model.

3.3 Benchmarks

Apart from a performance measure, we also need something to which to compare the performance. What's good and bad performance depends on the alternatives. The obvious benchmark is to use some kind of stock index, which is also the most common method used by professional broker companies. A broker who outperforms the index gets bonuses and the one who doesn't might risk his job. Mutual funds in particular are governed by these ways of thinking. Even if the individual traders wont lose their jobs, the managers know that their funds will lose customers if they start doing worse than the global stock index.

This is one of the reasons that so many mutual funds acting on one particular market perform almost identically.

For our purposes however, it's a reasonable benchmark since it compares the performance to a very available alternative: that of buying a mutual fund instead of doing the trading ourselves.

The following two kinds of indexes have been tried:

- The Swedish index *Generalindex.*
- An artificial index with equal parts for all the stocks available for the artificial trader.

No significant differences affecting the performance evaluation have been found between the two kinds of indexes and the official *Generalindex* has therefore been chosen the as benchmark.

3.4 Conclusions Regarding Performance Evaluation

The artificial trader's result is presented as annual profits together with the increase in index. The mean difference between these two figures constitutes the net performance for the trader. The performance is displayed in both tabular and graphical formats as shown in the table part of 3 and figure 1. The second last line of the table presents the annual profit above the index. The mean value of this entity is presented in the second rightmost column and represents a one-figure performance measure. However, the individual figures for each year should also be considered. The question whether the optimization and selection of trading algorithms should be based on the mean value or, for example, on the worst value for the investigated years, remains open.

4 Designing the Artificial Trader

The task of the Artificial Trader is to act on an artificial market with a large number of available stocks that vary in prices over time. The Trader has to execute the trading rule $T(t)$ at every time step and decide whether to buy or sell stocks. The sole aim of the Trader is to produce as high a profit as possible. Various aspects of the calculation of the performance were discussed in the previous section 3. The presentation and evaluation of the trading results is a major part of the ASTA system. In this section the general architecture of the developed system is described.

4.1 Basic Architecture

The architecture of ASTA is based on an object-oriented approach with two major objects; the Market and the Trader. The two objects have a number of attributes and operations that can be applied on the objects . The basic components and their relations are presented in figure 2.

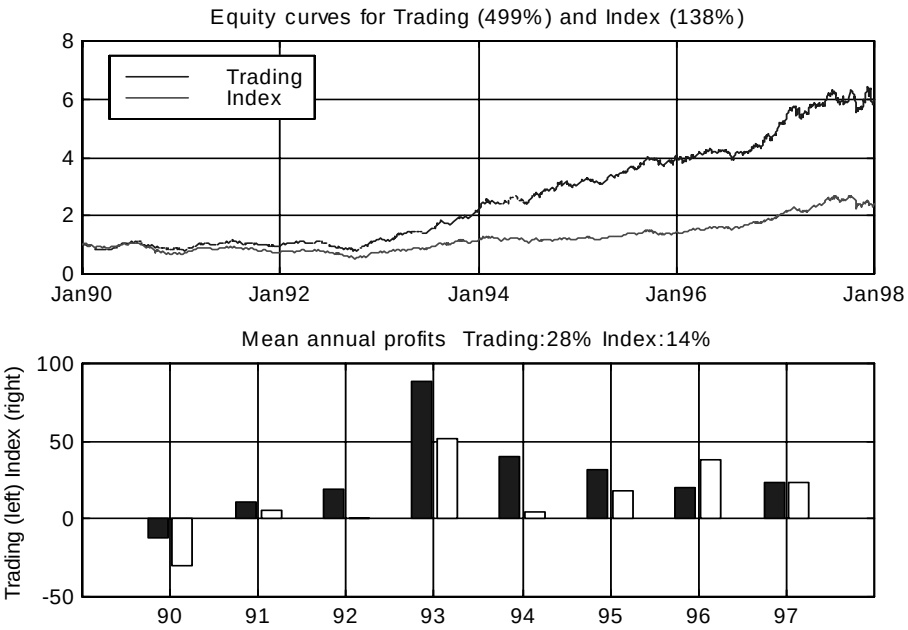

Figure 1: Presentation of performance for a (successful) trading strategy

4.1.1 The Market Object

The Market Object consists essentially of the total number of stocks that should participate in the trading simulation. A stock is defined by four time series; **Close**, **High**, **Low** and **Volume**. Each of these time series has a numeric value (or NaN in the case of not available values) for each date in the time period of interest. The basic operation on the Market Object is the simulation of changing prices as the date moves from start date to end date. The attribute **T** is updated by the **Step in time** operation applied to the Market Object.

4.1.2 The Trader Object

The Trader Object is more complex than the Market Object, as far as both attributes and allowed operations are concerned. The **Portfolio** attribute keeps track of the possession of stocks during the simulation. The **Cash** attribute is initialized to a certain value and thereafter is modified automatically as stocks are bought and sold. The **Buy rule** and **Sell rule** are the main attributes that affect the behavior of the Trader. They are expressed in a high-level language and may include calls to a large number of predefined Matlab functions that access the stock data in the Market Object. User-defined functions can also be called directly. The values of the **Buy rule** and **Sell rule** attributes can be set interactively to enable fast experimentation when developing trading algorithms.

The basic operations on the Trader Object are **Buy_recommendations** which evaluates the **Buy rule** and **Sell_recommendations** which evaluates the **Sell rule.** The operations compute vectors with buy and sell recommendations for all relevant stocks. These recommendations are then carried out by the **Buy** and **Sell** operation. The behavior of

the Trader can also be modified by a number of **Other parameters** that affect parameters such as minimum and maximum values for one individual trade.

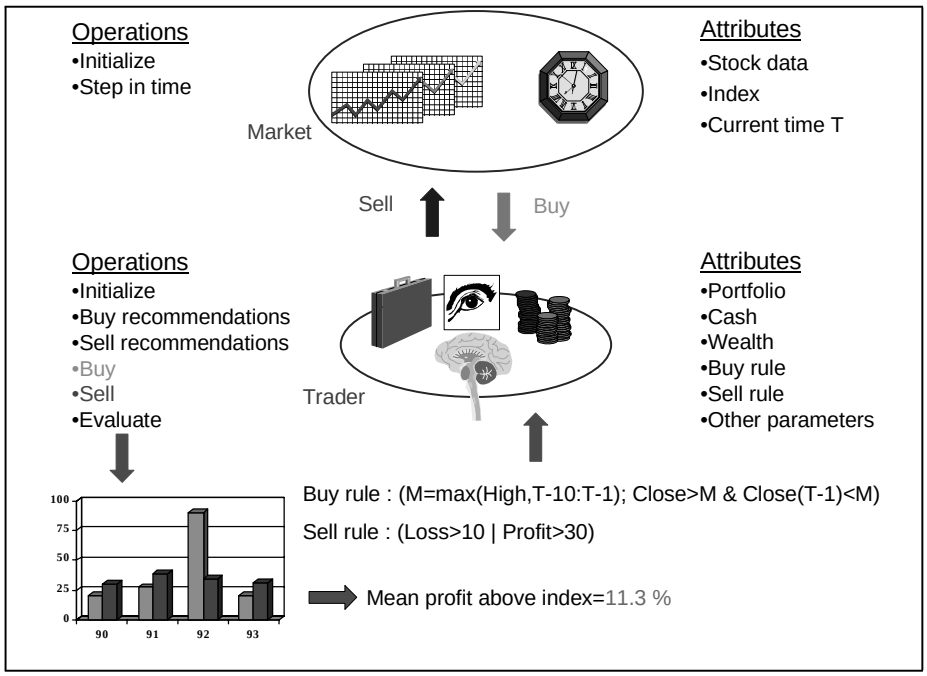

Figure 2: Basic components of the ASTA system

4.1.3 Other Parts of the System

The Market and Trader Objects have to be controlled by a support system that takes care of the following "meta" operations:

- Simulation.

  The **Step in time** operation has to be applied to the **Market** Object in a loop for the selected time period. For each time step, the Trader Object should also be activated. The following pseudo code describes the full ASTA system:

  Trader.Initialize
  Market.Initialize
  loop until **Market.T**≥EndDate
  s = Trader.Sell_Recommendations
  Trader.Sell(s) % Sell all stocks of type **s**
  s = Trader.Buy_Recommendations
  n = T.available_cash / length(s)
  Trader.Buy(s, n) % Buy **n** stocks of type **s**
  Market.Step in time
  end loop
  Trader.Evalute

- User interface

The end user should be able to set parameters such as the Trader's **Buy rule** and **Sell rule** and the desired time period for simulation. The computed performance of suggested trades etc., should be presented and also optionally printed out to the user after the completed simulation. In the case of using the system as an objective function in an optimization task, there must also be a "silent mode" included. The Artificial Trader should then be called as an ordinary objective function in the optimization program code.

4.2 Other Design Issues

We conclude the general description of the design with some specific issues that had to be taken into account in the design of the Artificial Trader.

4.2.1 When Is Data Available?

In a real trading situation on day T, Close, High, and Low for day T are not available. However, it's common that prediction algorithms assume that this data is available and can be included in the prediction of tomorrow's prices. Furthermore, it's often assumed that the Buy and Sell recommendations from the trading system can be executed using the close prices for day T. In reality this is of course hard to achieve, since the stock market is closed by the time the data for the current day becomes available, not to mention it having been transferred to a computerized trading system. ASTA can be configured to execute the simulated trades using either of the Following prices for buy and sell:

- Today's Close price
- Today's Mean price
- Tomorrow's Close price
- Tomorrow's Mean price

The data for day T is assumed to be available *for analysis* on day T.

4.2.2 How Many Stocks to Buy

The trading rule $T(s)$ does not contain any guide as to how many stocks or how big a portion of the cash should be invested in the particular stock that gets a *buy* signal. It's obvious that this affects the performance of a trading system and it's also clear that a *risk* estimate coupled with the trading signal could be of help. The present version of **ASTA** does not offer any sophisticated procedures for selection of the number of stocks to buy. The agent simply divides all available cash equally among the stocks that get a *buy* signal (while observing the upper and lower limits for one individual trade). When a stock gets a *sell* signal, the whole holding is sold.

4.2.3 Reinvesting Money

The Artificial Trader buys and sells stocks and hopefully increases the wealth during the simulated time period. In such a way there will be a cumulative effect, since old profits will be reinvested in future trades. It affects the performance evaluation as was discussed in more detail in Section 3.

Preventing the Artificial Trader from reinvesting the profits made along the way would remove the cumulative effect, but wouldn't provide a realistic situation since the real value of the working capital will be reduced due to inflation. The trader's effective capital by the end of a ten year period may be less than half, for example, of what it had been in the beginning. Therefore, we find it better to let the trader reinvest all the money to let the stock prices automatically compensate for the inflation.

4.2.4 Interest on Cash

The money not invested in stocks is assumed to yield interest in a bank account. The level of interest varies over time and follows the Swedish three-month-bond interest rate.

4.2.5 Unused buy and sell signals

One problem with the basic approach of evaluation by simulated trading is that a large portion of the buy signals will be neglected due to lack of money for the artificial trader. Even sell signals will be neglected simply because the stock that issue the signal is not in the traders portfolio. Tests show that more than 90% of the signals often are ignored in this way. This seriously affects the statistical basis of a performance analysis for the training algorithm. A possible way to attack the problem is a top level loop with multiple runs and randomization of entities such as:

- The starting date for trading
- The acceptance of buy signals
- The set of available stocks for trading

The present version of ASTA does not include any of the suggested improvements. Since the running time for one simulation of 10 years trading takes in the order of 30-60 seconds on a 266 MHz PC, the problem needs proper consideration before any further development along these lines are conducted.

5 Using ASTA

ASTA is available in two versions:

- As a Matlab function, fasta.m, that takes a **Buy rule** and **Sell rule** as in parameters and returns the performance for a selected time period.

- As a windows application to be run under Matlab. The **Buy rule** and **Sell rule** and all other settings are controlled interactively and the results are presented on the screen.

5.1 fasta

The purpose of the "batch" version of the program is to provide an objective function that can be used for parameter tuning or automated generation of trading rules. This report does not describe that part of the system more than by showing the following example of how it can be used:

Example 1 *:*

```
% load the workspace with stock data:
fasta('astasxg')
% Simulate trading with specified Buy and Sell rules for 1990-1995.
% The performance is returned in the p variable.
p = fasta('Close(T)>Close(T-1)','Profit>20 | Loss>10',[90 95])
```

5.2 wasta

Asta can also be run as an interactive windows application in the Matlab system. In this section examples from the interactive version are demonstrated.

To start ASTA, run the **wasta.m** function from the matlab environment. The following screen should be displayed:

In this picture, the user can set up parameters for a simulated trading. The stocks of interest are picked from a large database in Metastock format. Stocks from the Swedish stock market have been used in the presented runs. 32 major stocks with active trading for the years 1987-1997 have been selected. Other databases could easily be interfaced to the system.

The most interesting items are the lines "Buy rule" and "Sell Rule". This is the place where the trading algorithm is decided. The rules follow Matlab syntax and can include any of the large number of predefined functions or the users own functions with new algorithms. The simulation of trading starts by clicking on the "Run" button. The results are presented in tabular form and in graphs such as figure **??**.

The command button "Optimize" initiates a whole series of simulations with different values on symbolic parameters in the Buy and Sell rules. The name of the parameters is given in the "Parameter" text box and the values to be included in the multiple simulations are given in the "Values" text box. The results are presented in four graphs which are also written as eps files. Examples can be found in the section 8.

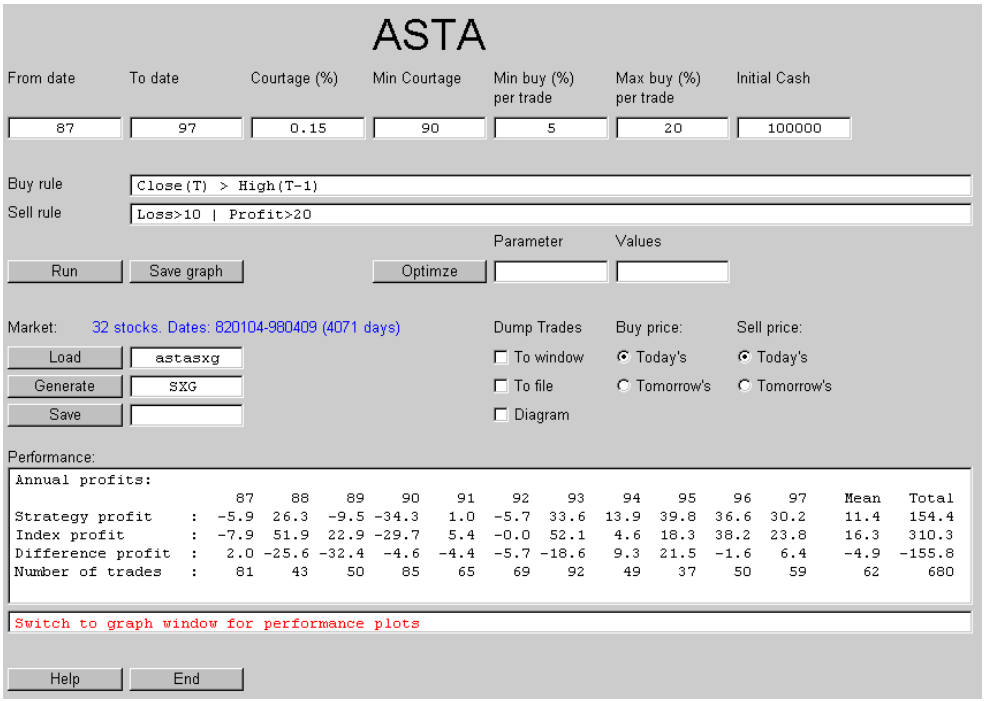

Figure 3: Screen layout for the windows version of ASTA

6 Developing Trading Algorithms with ASTA

One of the key goals of using ASTA has been to provide a solid and easy-to-use basis for development of trading algorithms. The "Trading rule" approach has been taken as described in Section 2.2. The algorithm should be formulated in a day-by-day structure, where buy and sell recommendations are produced daily by the algorithm. The decisions should be based on past stock data only. Fundamental data about the companies cannot be used in the present version of the system. It should also be emphasized that the "Trading rule" approach implies a decision-making system and not a modeling system. However, implemented algorithms may use modeling "internally" and base the buy and sell recommendations upon it.

The following goals and guidelines have been kept in mind while designing the development interface in ASTA:

- Fool proofness

  An "off-by-one-error" can seldom result in more drastic consequences than when developing prediction algorithms. Looking into the future, which is normally pretty hard to do, is in the case of prediction algorithms *surprisingly* easy to do, even unintentionally. In the case of indexing time series vectors however, it should not come as a surprise that finding an error in code segments, like the following one, is indeed difficult.

  $i = k - h + 1$

  $x(m) = y(k + 2 * (j - j) - 1)...$

Therefore, the access functions implemented in **ASTA** have been designed with a check for arguments that peep into the future. The global variable **T** is automatically updated by the simulation mechanism to point at the "current day". Attempts to address the stock time series before this point will automatically issue an error message. Example:

$y = Close(T-4:T)$ assigns the close prices for the current day and the 4 previous days to the variable y.

$y = Close(T:T+1)$ issues an error message when executed during the simulation.

The same check routine has been implemented in the higher level routines as well, thereby issuing clearer error messages pointing to the first place, where an illegal time reference is found.

- Ease of use

  Existing ASTA functions can be used interactively to create **Buy rules** and **Sell rules**. When the user has implemented a new trading algorithm the Matlab function can also be used in the **Buy rules** and **Sell rules.** It can easily be tried out interactively with different settings on parameters etc.

- High level functionality

  Predefined ASTA functions can perform operations such as moving averages and detection of crossings between time series. Other predefined functions compute derived entities from the stock data time series; gaussian volume, volatility, trend and ranks. These functions can be included in the user-defined algorithms and will simplify the coding considerably.

ASTA is equipped with a large number of predefined functions to be used interactively in **Buy rules** and **Sell rules.** They may also be used in user-defined functions. Before going into details of the available functions, some basic concepts will be described.

6.1 Global Variables

The variables listed in table 1 are dynamically set by the system and should normally not be changed by any user routines.

The variable **T** is central both when defining **Buy rules** and **Sell rules** and also when writing new functions for prediction. The idea is that the user never should have to worry about the time aspect, since it is automatically taken care of by the simulation framework. By using **T** as index, the "current" day will always be referenced. By using **T-1**, yesterday will be referenced etc. Attempts to use **T+1** will issue an error message, thus prohibiting the program from peeping into the future.

It is seldom necessary to use the variable **Stocks** when writing user-defined functions. However, it should be made clear that the variable is set to all stocks in the market when the system looks for stocks to buy (i.e. when the **Buy rule** is evaluated) and to all stocks in the Trader's portfolio when the system tries to find stocks to sell (i.e. when the **Sell rule** is evaluated). The function call **Close(T)** for example, returns a row vector with the

Name	Type	Size	Decription
T	numeric	1	Row pointer to "current" day
Stocks	numeric	$1xN$	Column pointers to allowed stocks. **Stocks** is set to all stocks in market when evaluting a **Buy rule** **Stocks** is set to all stocks in portfolio when evaluating a **Sell rule**
Market	struct	1	The **Market** object
Trader	struct	1	The **Trader** object
ST	struct	1	Parameters controlling the Trader's behavior

Table 1: Global variables in ASTA

close values for all stocks in the **Stocks** variable for the day **T** (the "current" day in the simulation).

6.2 Time Series Matrices

Most time series data within the ASTA system are enclosed in objects denoted *Time Series Matrices* and are abbreviated *TSM* in this documentation. They are ordinary 2-dimensional Matlab matrices where each row represents one date and each column represents one stock. The actual dates and actual stocks in a particular matrix vary, but the last row most often corresponds to the "current" day **T**. The columns normally correspond to the stocks in global **Stocks.**

The available data is the stock time series enclosed in the **Market** Object; Close, High, Low and Volume. These are the only features available when creating **Buy rules** and **Sell rules**, and also when creating new prediction algorithms. **Close, High, Low,** and **Volume** are also the names of the access functions used throughout the system.

Example 2 *: The call* ***Close(T-5:T)*** *returns a 6 row matrix where each column is assigned the close value for a stock in the global* ***Stocks*** *variable. Row 1 corresponds to day* ***T-5****, row 2 to day* ***T-4,*** *down to row 6, which corresponds to day* ***T****.*

A whole range of functions that accept TSM:s as arguments and/or generate it as output are included in the system. This design principle turns out to be convenient, since the matrix operations within Matlab can be used and makes the code very readable and effective. The Matlab function OHIGH in the following example detects Close values that cross a k day maximum of High from below.

Example 3 *:*

```
function b = OHIGH(k)
  C = Close(T);
  Cm1 = Close(T-1);
  M = max( High( T-k:T-1 )); % The call to High returns a k
                             % rows long Time Series Matrix
  b = C>M & Cm1<=M;
return
```

The function returns a binary row vector with a "1" in those columns where the corresponding stock fulfilled the condition. It is an example of the data type: *Indicator Vector.*

6.3 Indicator Vectors

An Indicator Vector is a single-row TSM. Sell and Buy rules should evaluate to binary Indicator Vectors that are used directly as sell and buy recommendations. Other Indicator Vectors such as the return value from the function calls **Profit** and **Trend1** contain real numbers as values.

In the previous section the function OHIGH returned an Indicator Vector with binary values, "0" or "1". A number of predefined functions return Indicator Vectors, and a user-defined trading algorithm or component should normally return a vector of this type. This enables such compositions of **Buy** rules as:

```
OHIGH(10) & ( Trend1>1 | Trend3>0.5 )
```
(10)

The **Buy** rule evaluates to a binary Indicator Vector with "1" for stocks where OHIGH(10) returned "1" and either the short trend is greater than 1%/day or the long trend is greater than 0.5%/day. Functions **Trend1** and **Trend3** are described below.

6.4 Predefined ASTA Functions

This section provides a list of all predefined functions that have been included to make it possible to express compound trading rules interactively (as **Buy rules** and **Sell rules**) and also to provide the developer of new algorithms with basic access functions as well as some useful high level functions.

The functions are divided into four categories:

1. Feature Functions.

   Feature Functions are primarily access functions to stock prices (**Close, High, Low**) and traded **Volume**. Numerous functions for commonly used derived entities, such as trend, gaussian volume, and volatility, are provided as well. A Feature Function always returns a TSM.

2. Indicator Functions.

   Indicator functions return an Indicator Vector most often used in the logical expressions that define the **Buy rules** and **Sell rules**. Examples are standard technical indicators, such as MACD and Stochastics. Other Indicator Functions implement a dynamic stop-loss handling and achieved profit or loss since a stock was bought.

3. Operator Functions.

   Operator functions perform a computation on one or several TSM:s and return another one. Examples are computation of moving averages, minimum values in time windows etc.

4. Utility Functions.

   Some functions are support functions that may be of use primarily when debugging new algorithms. Functions that extract stock names and actual dates are in this category.

IMPORTANT:

All functions of type 1, 2, and 3 return a TSM. Each column represents one stock in the global variable **Stocks. Stocks** is automatically set to all stocks in the market when the system looks for stocks to buy (i.e. when the **Buy rule** is evaluated) and it is set to all stocks in the Trader's portfolio when the system tries to find stocks to sell (i.e. when the **Sell rule** is evaluated). It is seldom necessary to use the **Stocks** variable explicitly in **Buy rules** or **Sell rules** or when writing user-defined functions.

6.5 General Parameters

This section describes some parameters, which are common to many of the predefined functions.

6.5.1 days

This parameter tells for which day or days the function should return values. In the case of **days** being a vector, the result is a TSM with the same number of rows as elements in the **days** parameter. Each row contains values as if computed on the corresponding day.

The default value for the **days** parameter is the global **T**, i.e. the current day.

Example 4 *: **Close(T)** returns a single row TSM with close prices for all the stocks in the global variable **Stocks**.*

Example 5 *: **Trend3(T-9:T)** returns a ten-row TSM with 5-day trends for all the stocks in the global variable **Stocks**.*

To simplify the description of functions below, the **days** parameter is not covered. The **Description** field in the tables refers to *each* row.

6.5.2 plot

Many functions have the optional **plot** argument. By setting this parameter to the global constant **PLOT**, the function is plotted versus time for all the selected stocks selected in a row. The exact contents of the plot depends on the function. User-defined functions can easily incorporate the plotting functionality, which is extremely useful during development and debugging.

Only one function at a time can have the **plot** parameter set to a non-zero value (the global constant **PLOT** equals to 1). The default value for the **plot** parameter is 0.

6.6 Feature Functions

Feature Functions are primarily access functions for obtaining stock prices (**Close, High, Low**) and traded **Volume**. Derived entities such as ranks and trends are also provided.

All the Feature Functions return a TSM with one row for each day in the **days** parameter and one column for each stock in the global **Stocks** variable. Table 2 lists all predefined Feature functions in **ASTA.**

Function	Parameters	Description
Close	**days, plot**	The close value
High	**days, plot**	The highest traded price during day day
Low	**days, plot**	The lowest traded price during the day
Volume	**days, plot**	The traded volume value
Rank1	**days, plot**	The 1-day rank value
Rank2	**days, plot**	The 2-day rank value
Rank3	**days, plot**	The 5-day rank value
Rank4	**days, plot**	The 20-day rank value
Trend1	**days, plot**	The 1-day trend (%/day)
Trend2	**days, plot**	The 2-day trend (%/day)
Trend3	**days, plot**	The 5-day trend (%/day)
Trend4	**days, plot**	The 20-day trend (%/day)
Gvol1	**days, plot**	The 2-day gaussian volume
Gvol2	**days, plot**	The 5-day gaussian volume
Gvol3	**days, plot**	The 10-day gaussian volume
Gvol4	**days, plot**	The 20-day gaussian volume
Volatility1	**days, plot**	The 2-day relative volatility
Volatility2	**days, plot**	The 5-day relative volatility
Volatility3	**days, plot**	The 10-day relative volatility
Volatility4	**days, plot**	The 20-day relative volatility

Table 2: Predefined feature functions

Some special concepts need to be further explained:

6.6.1 k-step Return

The k-step return $R_k^m(t)$ of a stock m with a price time series $Close^m(t)$ is defined as

$$R_k^m(t) = 100 \cdot \frac{Close^m(t) - Close^m(t-k)}{Close^m(t-k)}. \tag{11}$$

By setting k at different numbers we get measures indicating how much the stock has increased since its value k days ago. The Return values are not pre computed in **ASTA**. The related **Trend** functions described in the next section can often be used instead.

6.6.2 k-step Trend

The k-step trend $T_k(t)$ of a stock m is defined as

$$T_k^m(t) = \frac{1}{k} \cdot R_k^m(t). \tag{12}$$

By setting k at different numbers we get measures indicating how much the stock has increased per day since its value k days ago.

The four functions **Trend1**, **Trend2**, **Trend3**, and **Trend4** can be used to access four return measures for $k = 1, 2, 5$ and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

6.6.3 Rank

The k-step Ranks A_k^m for stocks $\{s_1, ..., s_N\}$ are computed by ranking the N stocks by the k-step returns R_k^m . The ranking orders are then normalized so the stock with the lowest R_k gets rank -0.5 and the stock with the highest R_k gets rank 0.5. The definition of the k-step Rank A_k^m for a stock s_m, belonging to a set of stocks. $\{s_1, ..., s_N\}$ can therefore be written as

$$A_k^m(t) = \frac{order(\ R_k^m(t),\ \ \{R_k^1(t), ..., R_k^N(t)\}\ ) - 1}{N - 1} - 0.5 \tag{13}$$

where the *order* function returns the ranking order of the first argument in the second argument, which is an ordered list. R_k^m is the k-step return (definition 11) computed for stock m.

The Rank for a stock is a measure seldom or never seen as input to Automatic Trading Systems. Since the ultimate goal is to "beat the market", i.e. to do better than the average stock, it seems reasonable to have them at least available as potential inputs. The rank concept is further analyzed in xxx.

The four functions **Rank1**, **Rank2**, **Rank3**, and **Rank4** can be used to access four rank measures for $k = 1, 2, 5$ and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

6.6.4 Gaussian Volume

The gaussian volume $V_n(t)$ is defined as

$$V_n(t) = \ (V(t) - m_V(t))/\sigma_V(t) \tag{14}$$

where $m_V(t)$ and $\sigma_V(t)$ are computed in a running window of length n. I.e.:

$$m_V(t) = \frac{1}{n} \sum_{i=1}^{n} V(t-i) \text{ and} \tag{15}$$

$$\sigma_V = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(V(t-i) - m_V(t))^2}. \tag{16}$$

V_n expresses by how many standard deviations the volume differs from its running mean. Therefore, it can be used as an indication of "abnormal" values caused by a sudden change in market interest for a particular stock. That's the reason why the sums in definition 15 and 16 don't include today's volume, i.e. $V(t)$. The four functions **Gvol1**, **Gvol2**, **Gvol3**, and **Gvol4** can be used to access four Gaussian volumes V_n for $n = 2, 5, 10$ and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

6.6.5 Relative Volatility

The Relative volatility $W_n(t)$ is here defined as

$$W_n(t) = 100 \cdot \frac{\sigma_C(t)}{m_C(t)} \tag{17}$$

where $m_C(t)$ and $\sigma_C(t)$ are computed on the time series Close in a running window of length n. I.e.:

$$m_C(t) = \frac{1}{n}\sum_{i=0}^{n-1} Close(t-i) \text{ and} \tag{18}$$

$$\sigma_C(t) = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n-1}(Close(t-i) - m_C(t))^2}. \tag{19}$$

Definition 17 is recognized as the "coefficient of variation" or "relative standard deviation" in statistics. Relative volatility W_n is preferred for ordinary volatility estimation by the standard deviation, because it provides a measure that can be compared over longer time periods and also between stocks with totally different price levels.

The four functions **Volatility1**, **Volatility2**, **Volatility3**, and **Volatility4** can be used to access four volatilities W_n for $n = 2, 5, 10$ and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

6.7 Indicator Functions

These functions return an Indicator Vector, i.e. a TSM with one row and one column for each stock in the global variable **Stocks**. Each column can be used in logical expressions to compose **Buy rules** and **Sell rules.** They can also be used with user-defined functions to create compound trading rules. Table 3 lists all predefined Indicator functions in **ASTA.**

Some Indicator Functions require further explanation:

Function name	Parameters	Description
Profit	-	% increase in **Close** since buy of stock
Loss	-	% decrease in **Close** since buy of stock
Stoploss	**S1, S2, P1**	Implements a dynamic stop-loss function: Stop loss is initially set to **S1** If **Profit**>**P1**, change the stop loss to **S2** **Stoploss** returns "1" iff the **Loss** is less than the dynamic set stop loss. This function keeps track of each stock separately.
Underlow	**L**	"1" iff **Close** crosses a **L** day min of **Low** from above
Overhigh	**L**	"1" iff **Close** crosses a **L** day max of **High** from below and **High(T-1)** is less than the same max
Days	**dys**	"1" iff day number is in vector dys (1:Monday...7:Sunday)
Stoch	**K,Ks,S**	Implementation of the technical indicator Stochastics.
Macd	**K,D,S**	Implementation of the technical indicator MACD

Table 3: Predefined indicator functions

6.7.1 Profit and Loss

The system automatically keeps track of the change in price of each stock since it has been bought. The **Profit** is defined as

$$\textbf{Profit}(t) = (Close(t) \ / \ \text{BuyPrice} - 1) * 100. \tag{20}$$

The **Loss** is sometimes a more convenient measure and is defined as

$$\textbf{Loss}(t) = (1 - Close(t) \ / \ \text{BuyPrice}) * 100. \tag{21}$$

BuyPrice is the average buy price of each stock in a portfolio. **Profit** and **Loss** are not defined for stocks outside the portfolio.

The functions are of great use when defining trading rules. For example, a simple **Sell rule** may be defined as

```
Loss>10 | Profit>20
```
(22)

The **Sell rule** evaluates to an Indicator Vector with "1" for stocks where either **Loss** returned a value less than 10 or **Profit** returned a value greater than 20. A more complex function based on the profit and loss concept is **Stoploss.**

6.7.2 Stoploss

An often mentioned "trading rule to live by" is that of never allowing a profitable trade to turn into a loss. This idea is implemented in the function **Stoploss**. Each stock gets a

stop-loss parameter of its own. This parameter is initialized to the value **S1** when a stock is bought. It is changed to **S2** when the **Profit** function registers more than **P1**% profit. Typical value for the parameters **P1, S1** and **S2** are 20%, 10%, and -5% respectively. These values gives the following behavior during trading:

1. The stock will be sold immediately if the price drops by more than 20% below purchase price, thus cutting the maximum loss.
2. If the price ever rises by 10% above the purchase price, a consecutive drop in price to 5% above purchase price will issue a sell signal. In this way, a profitable trade will never turn into a loss.

6.8 Operator Functions

These functions perform an operation on one or several TSM:s and return another TSM with transformed values.

All Operator Functions return a TSM with one row for each day in the **days** parameter and one column for each stock in the global **Stocks** variable. The parameter **x** (same applies to **x1** and **x2**) can be either a TSM or a string with the name of a function returning a TSM.

The parameter **days**:

- If **x** is a TSM, the **days** parameter should be row numbers in the matrix. The default value for **days** is the last row in the matrix**.**
- If **x** is a string, the **days** parameter should be a vector with day numbers such as **T**, **T-1** etc. The default value for **days** is **T.**

The Operator Function performs its operation on each of the lines in the **x** matrix and returns a TSM with the result for each row. In the following description of Operator Functions, the **d** variable is implicitly assumed to run through the full range defined by the **days** parameter. Each value on **d** results in one row in the output TSM.

Table 4 lists all predefined Operator functions in **ASTA.**

Function name	**Parameters**	**Description**
Minn	**x, L1, days, plot**	The **L1**-day-minimum value in the rows of **x**
Maxx	**x, L1, days, plot**	The **L1**-day-maximum value in **x**
Mav	**x, L1, days, plot**	The **L1**-day-moving av. for columns in **x**
Stdev	**x, L1, days, plot**	The **L1**-day-st.deviation for columns in **x**
Mavx	**x1,L1,x2,L2,A, days,plot**	"1" iff **Mav(x1,L1)** crosses **Mav(x2,L2)** from below. The **A** argument (optional) is the min required angle for the crossing.

Table 4: Predefined operator functions

6.8.1 Minn

Computes the minimum value in a L1 days long window backwards.
If x is a string, $Minn$ **is defined as:**$Minn \ = \ min(x[d - L1 + 1 : d])$
where the square brackets denote function invocation.

Example 6 $Minn('Close', 4, T)$

which will generate a 1 row TSM: $min(Close(T - 3 : T))$.

Example 7 $Minn('Close', 4, T - 1 : T)$

which will generate a 2 row TSM: $\begin{pmatrix} min(Close(T - 4 : T - 1)) \\ min(Close(T - 3 : T)) \end{pmatrix}$.

If x is a not a string, $Minn$ **is defined as:**$Minn \ = \ min(x(d - L1 + 1 : d, :))$.

Example 8 $Minn(Z, 4, 10)$ *where Z is a 10 rows long TSM.*

which will generate a 1 row TSM: $min(Z(7 : 10\ , :))$.

Example 9 $Minn(Z, 4, 9 : 10)$ *where Z is a 10 rows long TSM*

which will generate a 2 row long TSM: $\begin{pmatrix} min(Z(10 - 4 + 1 : 10\ , :)) \\ min(Z(10 - 4 + 1 : 10\ , :)) \end{pmatrix}$.

6.8.2 Maxx, Mav and Stdev

These functions have the same parameters and behavior as **Minn** above. Just substitute $min\ for$ max, *mean, and std.*

6.8.3 Mavx

Crossings between moving averages of various signals are often used in the definition of traditional technical indicators. The **Mavx** function supplies a convenient way to implement the detection of such crossings. The optional argument **A** makes it possible to specify the minimum angle for the crossing. Crossings at lower angles than the specified one will not generate a logical "1" at that point. **Mavx** can be used to detect both crossings from below and above by changing the order of **L1** and **L2** arguments.

Example 10 *Mavx('Close',2,'Close',20,45) returns a binary Indicator Vector with "1" iff a 2 day moving average of* ***Close*** *crosses a 20 day moving average of* ***Close*** *from below in an intersection angle no less than 45 degrees. The computation is performed "today" since the* ***days*** *parameter defaults to* ***T****.*

6.9 Utility Functions

These functions are support functions that may be of use primarily when debugging new algorithms. They can not be included in **Buy rules** and **Sell rules** since they don't return Time Series Matrices. Table 5 lists all predefined Indicator functions in **ASTA.**

Date	**days**	Vector with the real date (yymmdd) for **days**
Stockname	**s**	String with the name of stock **s.**

Table 5: Predefined utility functions

7 Examples

In this section, some examples of runs with different Buy rules and Sell rules are presented. In figure 4 the stochastics indicator is investigated. The Stochastics indicator is very popular in real trading and is defined as:

The results shown in figures 4 and 5 are quite stunning, with an average annual profit of 53.4% compared to the 16.3% achieved by the index. Notable is that the only negative result is for the year 1997, where the strategy only made 17.7% while the index increased 23.8%. To investigate the trades further, the single year 1993 is run separately in figure 6. The buy rule

$$\text{Stoch}(30,3,3,20,80,\text{PLOT})>0 \tag{23}$$

has the PLOT option enabled in the last argument. This enables generation of separate graphs with the signals from the Stoch function. In figure 7 the components of the Stochastics indicator are displayed. The two horizontal lines mark the buy level (80%) and the sell level (20%). When the "Oscillator %K" value passes these lines, a buy or sell signal is issued. To analyze the generated trades in even more detail, the "dump trades to window" button has been checked. All generated trades will be dumped in ascii form in the Matlab command window. The result is presented in figure 8.

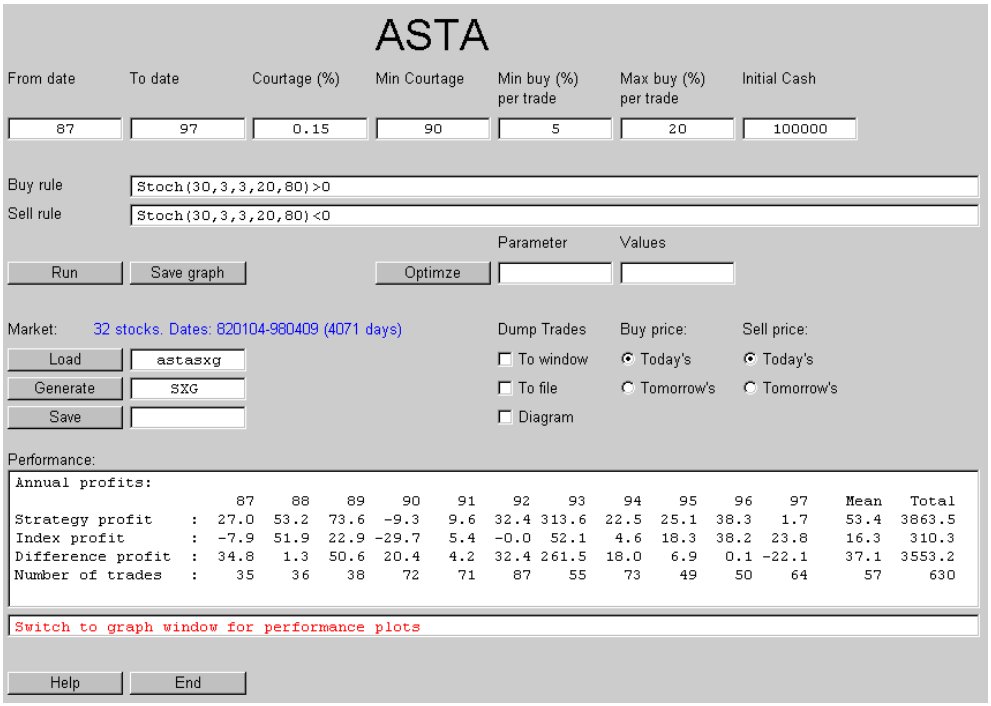

Figure 4: ASTA command window with Stochastics buy and sell rules

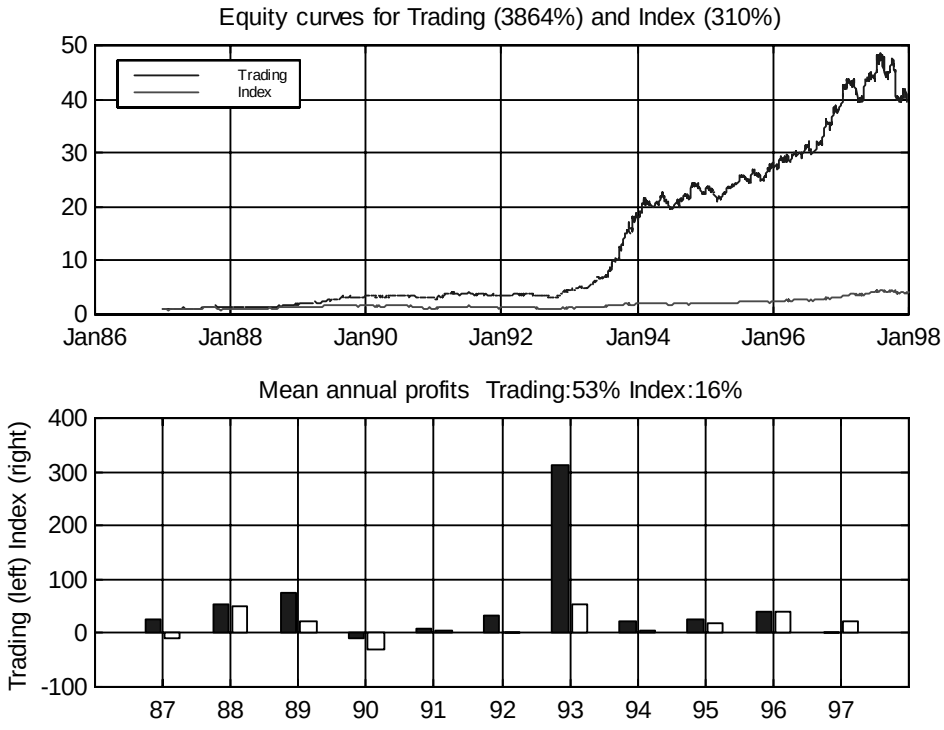

Figure 5: Performance for Stochastics indicator.

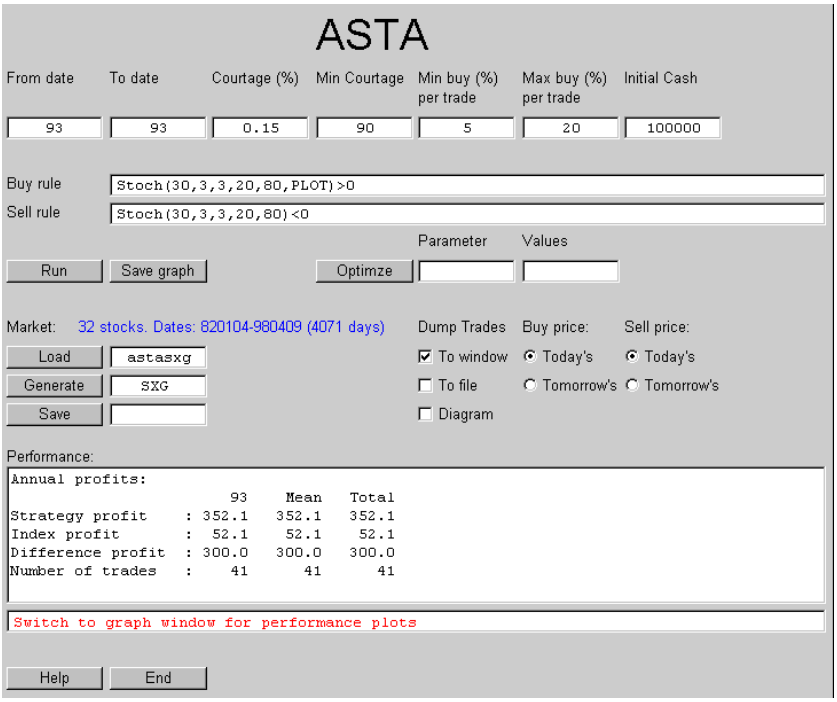

Figure 6: ASTA command window with Stochastics buy and sell rules. The PLOT option in the Buy rule generates separate plots with signals for each stock. The "Dump trades to window" check box generates a list of all trades in the Matlab command window.

8 Viewing the Trader as an Objective Function

Now that we have the simulation system working, we can start using it for many interesting tasks, some

of which are described in section 1.1. One of the proposed areas is to view the performance of the system as a function P of the trading rules. In the following we use the mean excess annual profit for a fixed time period (i.e. the profit minus the increase in index) as a one dimensional performance measure. Denoting this performance by p, we get

$$p = P(\textbf{Buy rule}, \textbf{Sell rule}). \tag{24}$$

Example 11 $p = P('Trend3 > 0.5 \ \& \ Gvol1 > 1.5', 'Loss > 10 \mid Profit > 20')$

The Matlab function *fasta.m* does implements the function P. It computes then mean excess annual profit given a **Buy rule** and a **Sell rule.** It is called from the interactive version of ASTA and can also be called from a user program for optimization or other purposes.

8.1 Optimization

The obvious wish to maximize the profit p can be tackled in two ways:

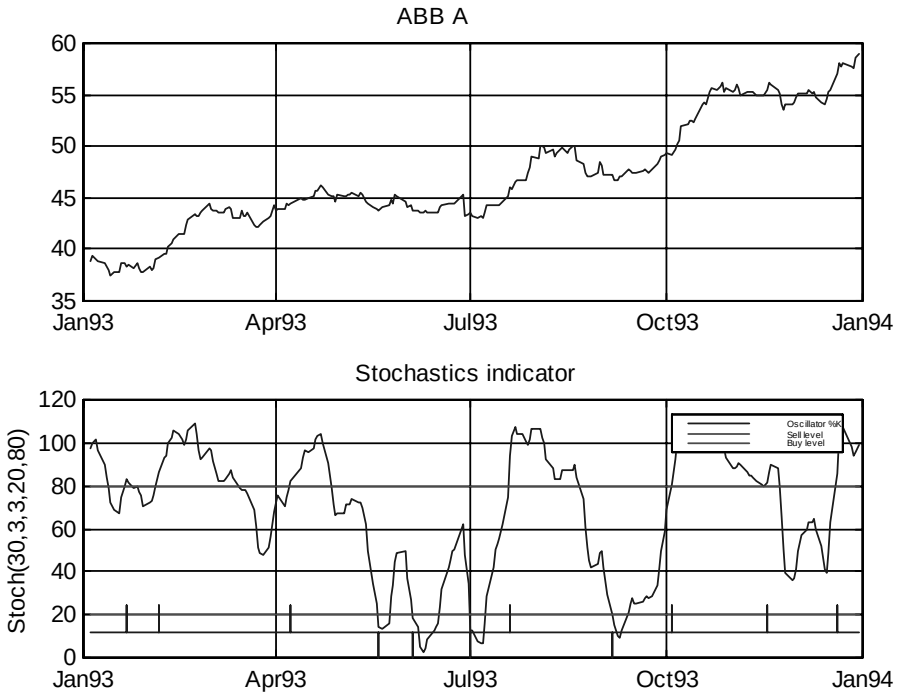

Figure 7: Stochastics buy and sell signals for the ABB stock. Upgoing bar denotes a buy signal and downgoing bar denotes a sell signal.

1. Parameterizing the **Buy rule** and **Sell rule**. I.e. introducing parameters within the rules. Example:

   Buy rule $= 'Close(T) > Maxx('High', Nhigh, T-1)'$

   Sell rule $= 'Loss > L \mid (Profit > P \ \& \ Close(T) < Close(T-1))'$

   The profit P can now be optimized with respect to the parameters $Nhigh, L$ and P.

2. Viewing the **Buy rule** and **Sell rule** as symbolic expressions. The optimization then turns into a search problem. The optimal **Buy rule** and **Sell rule** should be composed of atomic functions and operators from a set Ω. Example:

   $\Omega = \{Trend1, Trend2, Gvol1, Gvol2, Close(T),$ *Profit, Loss, ...*$, \&, \mid , >, < \}$

The profit P can now be optimized by combining elements in Ω to legal **Buy rules** and **Sell rules**.

In the general case, approach 1 is of course included in approach 2. The latter is however most naturally implemented in a genetic framework or with Inductive Logic Programming, while the former is a traditional parameter estimation problem. Genetic Programming applied to generation of trading rules can be found in [4]. Some experiments along the first approach have been conducted. One example is shown in figure 9, where the buy and sell rules from the example above are optimized with respect to the $Nhigh$ parameter. The graphs are automatically generated by the "Optimize" function in the ASTA system. The results are placed on a number of eps files of which some have been included in figure 10.

Trade	Date	No.	Stock	Price	Total Profit (%)		Wealth	Cash
1	930105	77	21 Skandia	85.27	6566.05			93343.95
2	930105	79	27 Sydkraft C	82.60	6525.22			86728.73
3	930105	1535	31 Allgon B	4.28	6574.72		99730.00	80064.01
4	930107	375	20 SHB A	26.34	9875.96			70119.50
5	930107	329	32 Nokia A	30.00	9870.00		99585.62	60159.50
6	930108	349	10 Hennes & Mauritz	28.20	9841.80			50260.08
7	930108	323	25 SSAB A	30.50	9851.50		99860.44	40318.58
8	930115	431	7 Avesta Sheffield	22.68	9776.63			30528.47
9	930115	351	20 SHB A	27.84	9771.77		100374.14	20666.71
10	930121	258	2 ABB A	38.40	9907.20			10702.32
11	930121	64	14 Kinnevik B	81.00	5184.00		NaN	5428.32
12	930127	54	21 Skandia	97.46	5262.68		NaN	84.48
13	930302	-64	14 Kinnevik B	79.00	-5056.00	-2.47	128010.13	5051.23
14	930322	-131	21 Skandia	93.71	-12275.62	3.78	131216.32	17260.75
15	930323	-431	7 Avesta Sheffield	23.16	-9980.37	2.08	NaN	27165.09
16	930325	-323	25 SSAB A	33.75	-10901.25	10.66	132663.45	37991.32
17	930329	4897	31 Allgon B	5.45	26675.18		132626.85	11247.39
18	930406	2031	31 Allgon B	5.51	11186.34		132246.33	1.77
19	930519	-258	2 ABB A	43.70	-11274.60	13.80	164709.29	11186.39
20	930525	113	22 Skanska B	97.50	11017.50		NaN	94.54
21	930608	-329	32 Nokia A	46.25	-15216.25	54.17	187009.69	15221.10
22	930610	154	27 Sydkraft C	97.95	15083.58		184210.74	54.44
23	930621	-349	10 Hennes & Mauritz	38.00	-13262.00	34.75		13226.55
24	930621	132	22 Skanska B	99.00	13068.00		185878.28	68.55
25	930716	-233	27 Sydkraft C	93.56	-21799.15	0.88	197758.60	21778.13
26	930719	308	7 Avesta Sheffield	34.97	10770.39		197540.39	10922.70
27	930720	235	2 ABB A	46.00	10810.00		199141.88	30.11
28	930906	-235	2 ABB A	47.20	-11092.00	2.61	273799.16	11032.41
29	930916	-308	7 Avesta Sheffield	36.86	-11352.70	5.41	272928.45	22322.39
30	930920	227	9 Ericsson B	97.83	22207.73		272118.39	33.78
31	930921	-245	22 Skanska B	138.00	-33810.00	40.37	270682.53	33753.80
32	930922	-726	20 SHB A	95.00	-68970.00	251.03	269365.98	102627.47
33	930923	672	32 Nokia A	80.00	53760.00		270719.82	48798.81
34	930927	562	10 Hennes & Mauritz	43.20	24278.40		282121.66	24450.66
35	931004	495	2 ABB A	49.20	24354.00		309655.25	42.43
36	931108	-227	9 Ericsson B	102.87	-23350.81	5.15	NaN	23303.54
37	931111	136	21 Skandia	170.55	23194.39		429195.43	42.48
38	931126	-672	32 Nokia A	100.50	-67536.00	25.62	399726.79	67477.30
39	931203	764	30 Volvo B	88.20	67384.19		387109.87	85.89
40	931215	-136	21 Skandia	159.30	-21664.94	-6.59	434649.03	21661.02

Figure 8: Dump of generated trades from Stochastics trading rules for the year 1993.

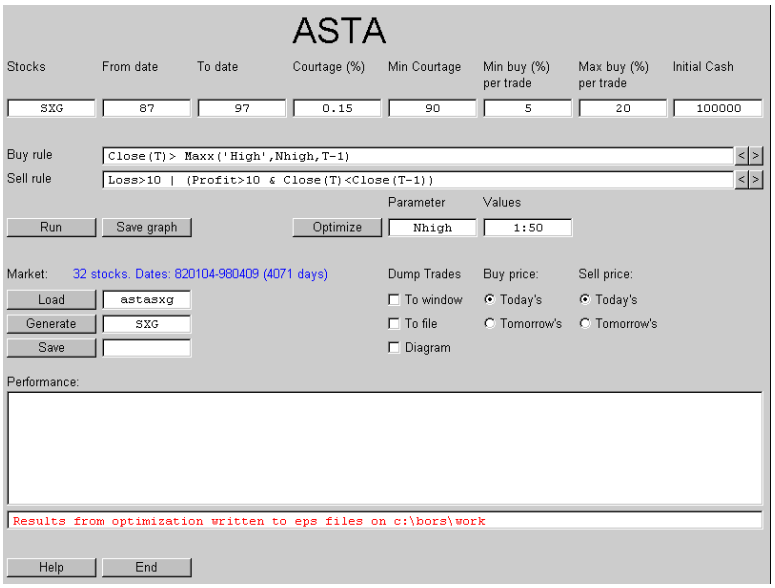

Figure 9: Optimization of the $Nhigh$ parameter. Buy rule: '$Close(T) > Maxx('High', Nhigh, T-1)$'. Sell rule: $Loss > L \mid (Profit > P \ \& \ Close(T) < Close(T-1))$'

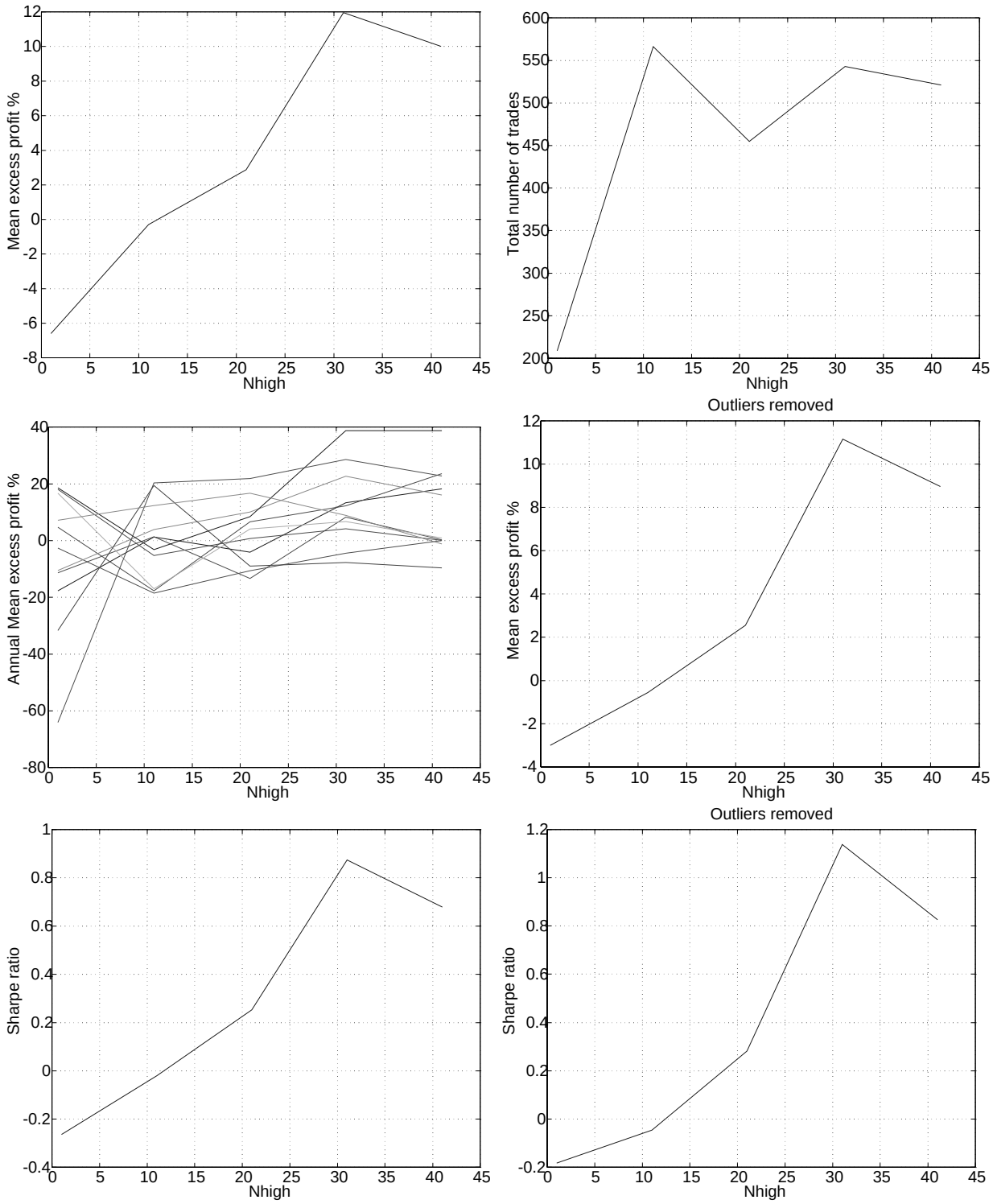

Figure 10: Trading results as a function of the Nhigh parameter. Stocks: SXG. Years:87-97. Buy rule: Close(T)>Maxx('High',Nhigh,T-1). Sell rule: Loss>10 | (Profit>20 & Close(T)<Close(T-1))

It must be emphasized that optimization of the performance is a multi dimensional parameter estimation problem. The graphs presented in this report show the profit P as a function of *one* of the involved parameters while the rest of the parameters are fixed. The main purpose is to illustrate the possibilities and problems involved in even a one dimensional optimization.

From this summary graph (top left) for the entire training period we can deduce that the highest profit is achieved for a $Nhigh$ somewhere between 10 and 15. However, looking at the data at a higher resolution reveals a more complicated situation. In the top right diagram the same relation is plotted with one curve for each year in the training data set. From this we can learn at least two important points:

1. The spread between individual years is very high.
2. The location of the maximum is not obvious.

This behavior of data has shown typical for most variables investigated. The profit can not be described as a function of measurable variables without introducing a dominant noise term in the function. It actually looks as if the underlying process generating the data is different for each year.

Let us view the annual excess profit as a stochastic variable $P[\theta]$ where θ stands for one particular setting of the parameters that affect the profit. In the shown example, θ is the $Nhigh$ parameter. $P[\theta]$ has been sampled once per year during the eleven years in the data set; $\{P_1[\theta], P_2[\theta], P_3[\theta], P_4[\theta], P_5[\theta], P_6[\theta], P_7[\theta], P_8[\theta], P_9[\theta], P_{10}[\theta], P_{11}[\theta]\}$. Viewed this way, it is clear that the task of tuning the parameter θ to find the "maximum" profit P is not well defined. P is a stochastic variable and consequently has no "maximum". It has a probability distribution with an expectation value E and a variance V. It's important to realize that tuning θ in order to maximize $E[P[\theta]]$ is just one of the available options. Maximizing $E[P[\theta]]$ gives the highest *mean* performance. Another possibility is to maximize the lower limit of a confidence interval. Since the risk factor is always a major concern in investments, and since the spread between individual years obviously can be very high, this sounds like a promising idea. A lower limit P_{low} for a confidence interval could be created:

$$P_{low} = E[P[\theta]] - \sqrt{V[P[\theta]]} \tag{25}$$

where $V[P]$ is the variance of the stochastic variable P. Yet another possibility is to use the Sharpe Ratio SR which expresses the excess return in units of its standard deviation as

$$SR = \frac{E[P[\theta]]}{\sqrt{V[P[\theta]]}}. \tag{26}$$

The Sharpe Ratio is normally used to *evaluate* the performance of a trading strategy (Sharpe [**?**],[**?**]). Choey and Weigend [3] however suggest to use the Sharpe Ratio as objective function in portfolio optimization and derive a learning algorithm for artificial neural networks. Due to the high noise level in the data we have added a modified Sharpe Ratio where the outliers have been removed. The largest and smallest $P[\theta]$ for each θ has been removed before the expectation value and standard deviation are computed. An

optimization of one of the parameters in the Stochastics indicator will serve as an example. We set up an optimization with buy and sell rules according to

Buy rule = '$Stoch(30, 3, 3, Sellevel, 80)$'

Sell rule = '$Stoch(30, 3, 3, Sellevel, 80)$'

The *Sellevel* parameter controls the position of the lower horizontal line (refer to figure 7) that generates sell signals in the Stochastics indicator. The excess profit is shown as a function of the *Sellevel* parameter in the top diagrams of figure 11. Unfortunately the function is still very bad behaved, and a clear maximum can still not be observed. By comparing the two top diagrams it is however clear that the mean profit is totally dominated by the results from *one* of the years (1993). This carries over to the *Sharpe Ratio* which has the same overall shape as the two profit diagram. The *Median based Sharpe Ratio* avoids the extreme values in the "outliers" and produce a more stable curve than the ordinary *Sharpe Ratio.*

A similar analysis of the effect of varying the *Buylevel* parameter is presented in figure 12. The function is also in this case very noisy and varies greatly from year to year. The need to improve the evaluation procedure with methods such as those described in section 4.2.5 becomes more and more apparent.

9 Results and further development

The presented system provides a powerful tool for development and evaluation of trading algorithms. Parameter settings can be tested and data screening can easily be performed interactively. As was mentioned in section 1.1, one of the reasons the ASTA system was developed was to act as an objective function when tuning parameters in the models or when finding the general structure of trading rules for example within a generic framework. This track can be examined considerably. It is also possible to get the trades dumped out of the system and use them as training data in traditional or novel classification methods.

The dangers with "data snooping" got highlighted by breaking down the performance measures into shorter intervals. The inherent uncertainty resulting from the noisy processes involved however calls for more computer intensive simulation schemes in order to achieve statistically significant performance measures.

References

[1] A. Atiya. Design of time-variable stop losses and profit objectives using neural networks. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 76–83, Singapore, 1997. World Scientific.

[2] Y. Bengio. Training a neural network with a financial criterion rather than a prediction criterion. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 36–48, Singapore, 1997. World Scientific.

[3] M. Choey and A. S. Weigend. Nonlinear trading models through Sharpe Ratio maximization. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 3–22, Singapore, 1997. World Scientific.

[4] A. N. Edmonds, D. Burkhardt, and O. Adjei. Genetic programming of fuzzy logic production rules with application to financial trading. In A.-P. Refenes, Y. Abu-Mostafa, J. Moody, and A. Welgend, editors, *Neural Networks in Financial Engineering, Proc. of the 3rd Int. Conf. on Neural Networks in the Capital Markets*, Progress in Neural Processing, pages 179–188, Singapore, 1996. World Scientific.

[5] G.Leitch and J.Tanner. Economic forecast evaluation: Profit versus the conventional error measures. *The American Economic Review*, pages 580–590, 1991.

[6] T. Hellström and K. Holmström. Predicting the Stock Market. Technical Report IMa-TOM-1997-07, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[7] J. E. Moody. Shooting craps in search of an optimal strategy for training connectionist pattern classifiers. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 NIPS Conference*, pages 847–854, San Mateo , CA, 1992. Morgan Kaufmann Publishers.

[8] J. E. Moody and L. Z. Wu. Optimization of trading systems and portfolios. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 23–35, Singapore, 1997. World Scientific.

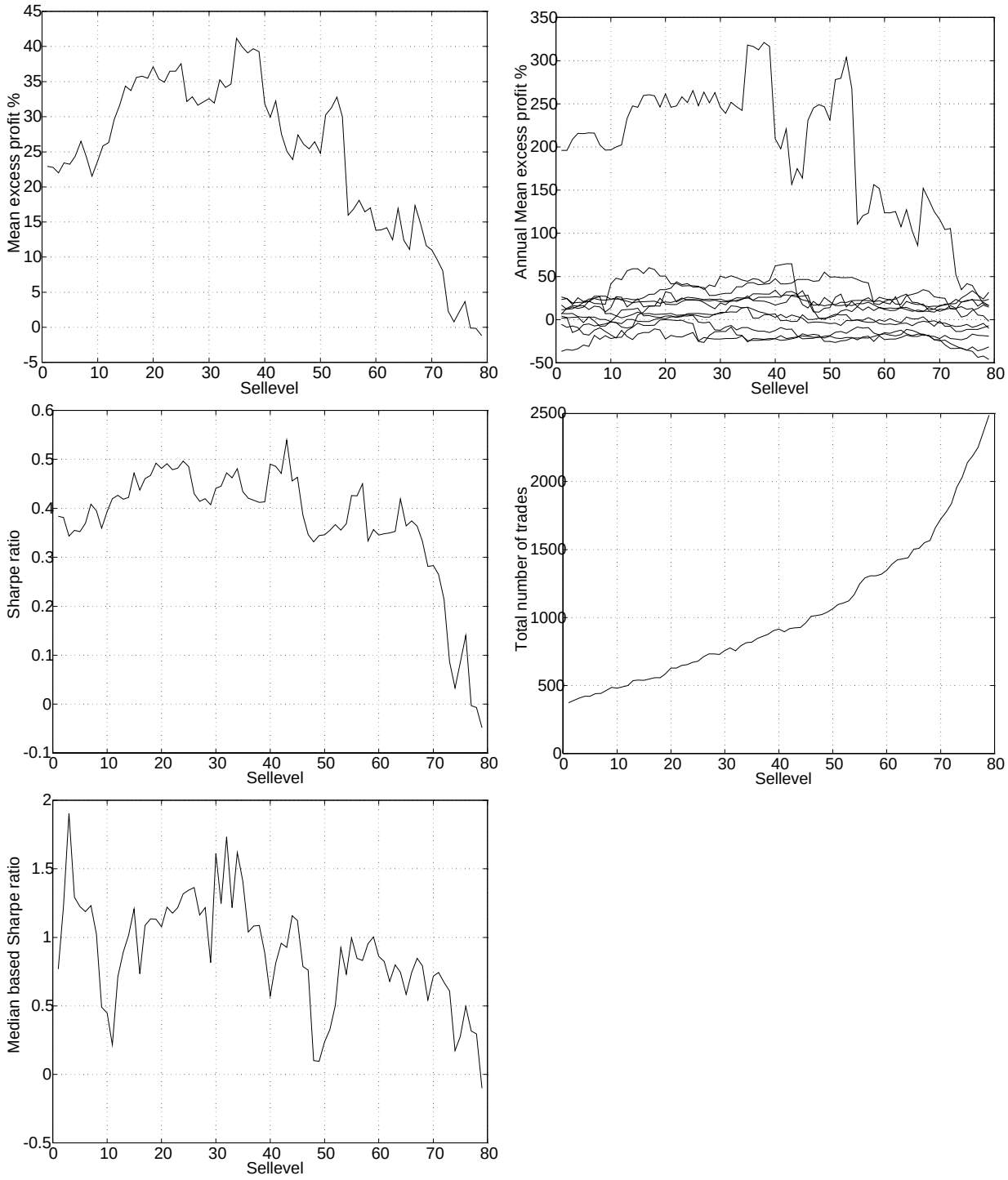

Figure 11: Trading results as a function of the Sellevel parameter. Stocks: SXG. Years:87-97. Buy rule: Stoch(30,3,3,Sellevel,80)>0. Sell rule: Stoch(30,3,3,Sellevel,80)<0

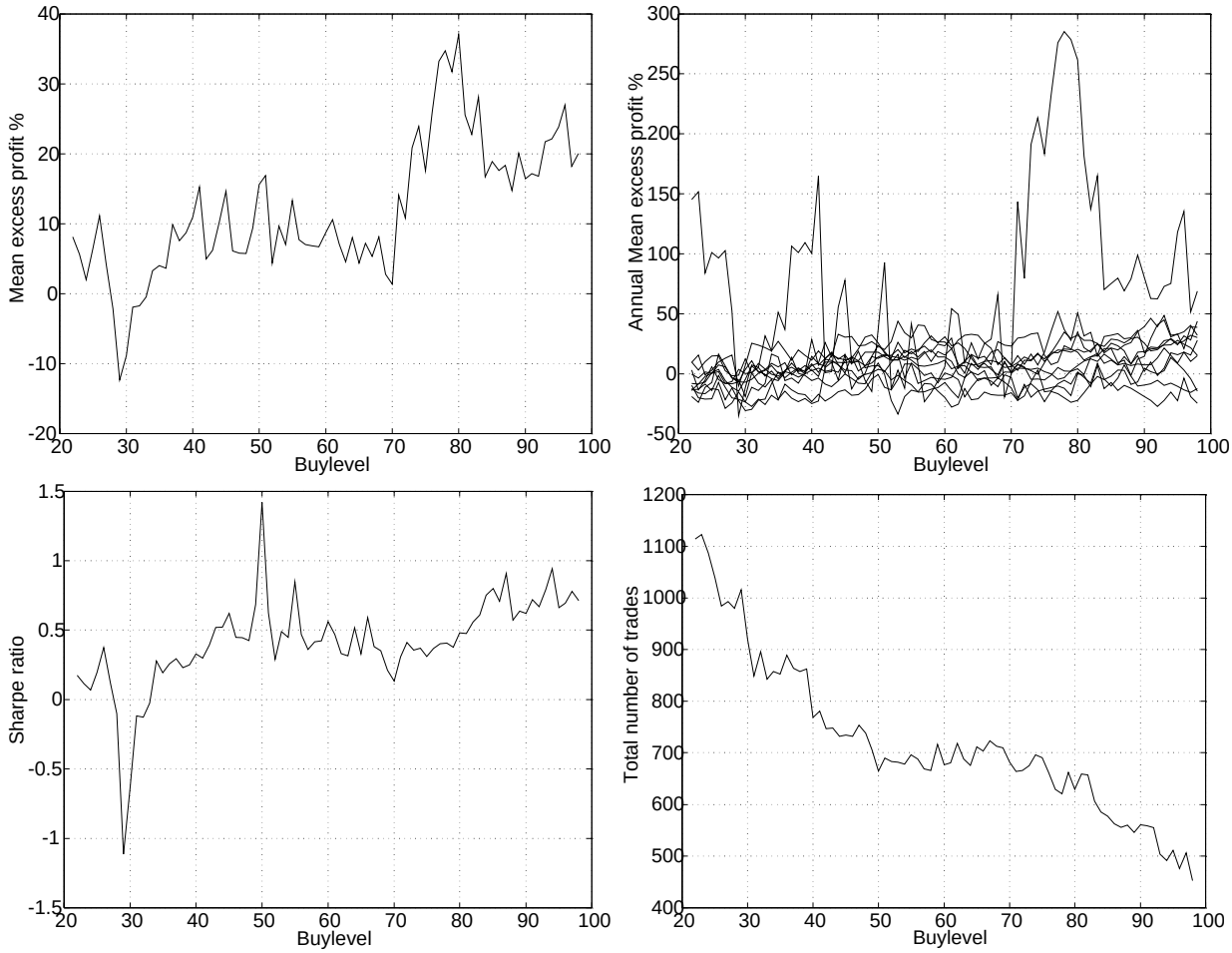

Figure 12: Trading results as a function of the Buylevel parameter. Stocks: SXG. Years:87-97. Buy rule: Stoch(30,3,3,20,Buylevel)>0. Sell rule: Stoch(30,3,3,20,Buylevel)<0