KTH Computer Science
and Communication

Algorithmic trading using MACD signals

FALK ANDREAS
MOBERG JOHANNES

Bachelor's Thesis at CSC
Supervisor: Alexander Kozlov
Examiner: Örjan Ekeberg

Abstract

Todays stock market is dominated by algorithmic trading either as helpful tool for trading decisions or as a fully automatic trader. We test how a fully automated trading algorithm using MACD signals as indicators perform on historical stock data. The purpose of this essay is to see how a simple algorithm performs and get a better understanding of economical forecasting.

Referat

Dagens aktiemarknad domineras av algoritmisk handel, både som ett hjälpmedel till handelsbeslut och som en fullt automatisk aktiehandlare. Vi har testat hur en fullt automatiserad handels algoritm som använder sig av MACD signaler presterar över historiskt handelsdata. Syftet med rapporten är se hur en enkel algoritm presterar och att få bättre förståelse för ekonomiska förhandsberäkningar.

Contents

Chapter 1

Preliminaries

1.1 Statement of collaboration

This work is a collaboration between Andreas Falk and Johannes Moberg as part of their bachelors degree at KTH. All parts have been done in collaboration and both individuals takes the same amount of credit for each part of the thesis.

Chapter 2

Introduction

Many studies has been made about different technical analysis and trading strategies. In this study we want to examine how a technical analysis indicator called moving average convergence/divergence analysis would have performed over a ten year span looking at a small set of stocks on the stockholm stock exchange, now part of the NASDAQ OMX Nordic. It seems the MACD has been tested in some academical reports but not as much as one would think taken how popular it is in real trading [1].

2.1 Background

2.1.1 Algorithmic trading

Algorithmic trading refers to all kind of trading using algorithms to automate all or parts of the trading procedure. Algorithmic trading has changed the way to see economical trading and is one of the fastest paradigm shifts in the market the past 20 years. In many ways it has completely revolutionized trading and extended the trading opportunities outside the boundaries of the capabilities of human calculation and decision making utilizing computers raw computational and data storage powers [2]. Algoritmic trading utilizes signals as input to the algorithms which is used to decide if the trader should for example place an order or close existing ones. Signals can be generated from data over long periods or very recent fluctuations in a stock or the market as a whole. Common parameters to generate signals are open and closing prices, total volume and in low latency trading short spanned timing signals.

2.1.2 Technical analysis

Technical analysis is the study of prices and trends in a freely traded market with the goal to make a profit from taken trading decisions. The main focus in technical analysis is to locate the direction of a found trend and act upon it in to maximize total profit. Trends are commonly divided in to uptrends, a rising price trend, downtrends, a shrinking price thrend and sideways trends were the trends tend to go rapidly up and down over a period of time but the average value stays the same [3].

2.1.3 Moving average convergence/divergence

Moving average convergence/divergence (MACD) as a technical analysis indicator was invented in the late nineteen seventies by Gerald Appel [4] and is one of the most effective yet simple trend following momentum based indicators to date. MACD is used to foretell the strength, change, direction, momentum and duration of a stocks price trend and was specifically designed to detect changes in trends at a relativley early stage.

It uses three signal as trading indicators calculated from historical price data. The three signals, or signal lines, that the MACD is built upon are the MACD line, the signal line (average line) and the difference line (divergence line). The MACD line is computed as the difference between a short periods and a somewhat longer periods exponential moving average (EMA), in simpler words it compares a short momentum to a slow momentum. We subtract the slow periods EMA from the fast EMA and plot it as the MACD line. The signal line is then computed as the EMA of the MACD-line and plotted alongside. The bar graph for the same time series is the divergence or convergence and shows the difference between the MACD line and signal line. Because the MACD takes in consideration both a fast and a slow momentum signal it becomes a mixture of a trend following indicator and a momentum based indicator also reducing some of the lagging indicators inherited by pure trend following indicators where the indicators changes when the market does as a whole.

To calculate our MACD we use a slow period 35-day-EMA, a fast period 5-day-EMA and a 5-day-EMA computed from the 35 and 15-day EMA. The exponential moving average of n days is calculated as:

$$EMA(n)_t = (\frac{2}{n+1})(P_t - EMA_{t-1}) + EMA_{t-1} \tag{2.1}$$

where P_t is the closing value of a stock on day t and n is the number of periods for calculating the EMA. The term $(\frac{2}{n+1)}$ is the weight of the day giving more significane to more recent data. The initial EMA is the n-day simple moving average of the series.

MACD is then computed as:

$$MACD_t = EMA(s)_t - EMA(l)_t \tag{2.2}$$

The last thing to compute is the 5-day-EMA, also called the signal line, which is simply the EMA of the MACD. The resulting set of indicators is commonly refered to as MACD(5,35,5).

The main purpose of the MACD-line and the signal-line is to use them as indicators to when to buy or when to sell a stock. The plot below shows us the lines for the HiQ stock during five months in 2014.

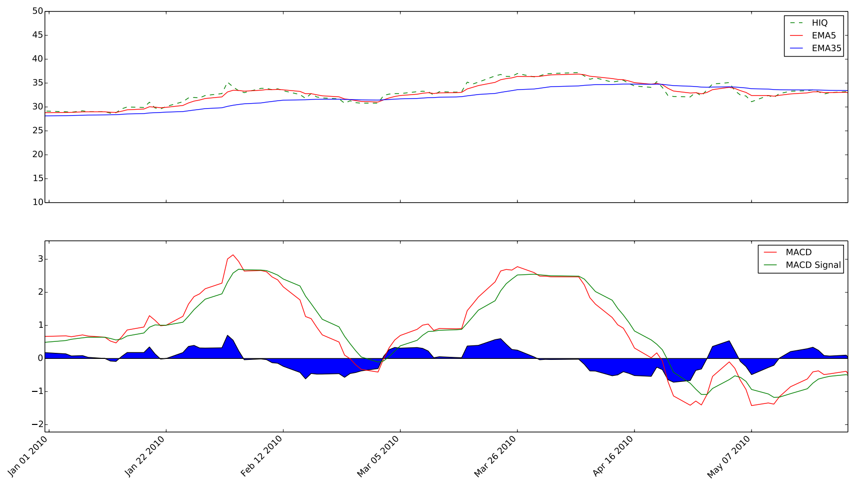

Figure 2.1. MACD(5,35,5) of the HiQ stock over five months

The main things that the MACD shows when plotted are the crossovers, convergences and divergences of the different lines. Convergence occurs when two lines move towards each other, divergence when they move away and crossovers when two lines pass each other. In the bottom part of figure 2.1 we have the MACD line plotted as the red line, the signal line as the green line and the difference between them as the blue histogram. The black horizontal line is called the center or zero line.

The most important signal is the singal line crossover occuring when the MACD line (red line) crosses the signal line (green line). The standard way to interpretate this is to buy when the MACD line crosses the signal line on its way up and to sell when it crosses on its way down. Crossovers are easy to spot in the histogram showing up when the bar is neither negative nor positve.
The second most important signal is the zero line crossover which takes place when the MACD line crosses the black horizontal line. At the crossing the

$$EMA(5) - EMA(35) = 0$$

and an upwards crossing indicates a time to buy and downwards a time to sell. This signal does, however, give less confirmation of the momentum.

2.1.4 Backtesting

Backtesting is a way to test newly inventeded trading algorithms and strategies over existing historical time series data sets. By using real data you can try to evalute how an algorithm or trading stratgey will perform in the future based on how it performed in the past. As in most fields where you try to predict the future false results may arise. As the algorithms are based on historical data they they are built upon the preliminaries that they will continue to perform good in the future which is not always the case as the market might change dramatically over a night. People also tend to overfit their implementations making them fit the time series used given a false picture of how they will perform in real action.

Chapter 3

Problem statement

This thesis will investigate what performance can be expected from a completely automated trading system when using MACD to generate buy and sell signals when having 100,000 SEK as initial investment capital.

The algorithm will be executed on daily summarized end-of-day data that contains opening price, closing price, low price, high price and the volume traded. The MACD signal generator will use the periods 5 days for the fast EMA, 35 days for the slow EMA and 5 days for the MACD signal.

Chapter 4

Methodology

In this part of the thesis it will be explained in detail how the benchmarking has been implemented and how some problems have been evaded to attempt to provide a realistic environment.

4.1 Backtesting

To evaluate what kind of performance that can be expected from the algorithm it will be simulated, or backtested on historical data.

This simulation is done by feeding the algorithm as one would do in a live environment but instead of using current data, historical data is used instead. When the algorithm generates a signal a virtual order is created that are later filled using a slippage model that is described in section 4.4.

Since feeding the algorithm in this way is the same as it would be fed in a live environment this should not introduce any changes to a live environment.

4.2 Data

The data used for the benchmarking was exported from Nasdaq OMX Nordics historical prices service on their website [5]. These exports are missing various data points but the only point that is relevant to this thesis is the volume. The volume was set to zero on the days where it was missing which forces us to make no trades on those days.

Since the data is exported from the stock market provider the data there is should be accurate.

4.3 Security selection

In an attempt to keep the number of parameters down a decision was made to choose one security exchange; NASDAQ OMX Nordic; and to pick one market index; OMX_NORDIC_MID_CAP_SEK_PI; from which to pick securities from. From this market index 5 securities has been randomly selected.

Coprorations can perform certain corporate actions on their stock that directly affect the stock price. Since the algorithm is completely dependent on the stock price it was decided to filter the selection from any securities that had either

- a stock split or
- a stock merger

In order to ensure that there was enough data the selection was also filtered from securities that did not have data for the full 10 years.

Using randomly selected securities is important to avoid selection bias and the filtering helps to minimize the number of variables in play as well as minimizing the amount of data mangling that needs to be done.

4.4 Slippage model

The slippage model is used to calculate the impact of the virtual orders on the market during the backtesting. The more realistic the model is the more realistic the simulation will be. For this thesis a slippage model was chosen that

- limits the volume available to trade relative to the total volume actually traded during that day and
- adjusts the final price of an order relative to the volume of the order

For this thesis it was decided to limit the available volume to be traded to 25% of the total traded volume on a single day and to have 0.1 as slippage constant. Thus the slippage s is calculated with Equation 4.1

$$s(v_{order}) = (\frac{v_{order}}{v_{total}})^2 * 0.1 \tag{4.1}$$

Both the volume limit and the slippage constant are fairly arbitrary however it is the default constants in the underlying libraries used when implementing the benchmark. The authors could not find any relevant research for better constants so it was decided that these values were good enough. Investigating better values for these constants might be an interesting topic for a future thesis.

4.5 Commission model

In Sweden most brokers use a per trade model with a minimum brokerage fee. For the chosen initial investing capital of 100,000 SEK the fees are around 0.085% of the total value of a order or a minimum of 99 SEK per order. Due to this a fixed commission model with a cost of 99 SEK was implemented for this thesis.

Since the value of the trades that are performed in this benchmark is relatively low this commission model provides a near perfect estimation.

4.6 Benchmarking

To be able to compare the returns of the algorithm (the value of the virtual portfolio) against the returns of the security and and index the different series of data has been normalized against their initial value.

For this thesis it is only interesting to compare the changes in value relative to the initial value. Because of this normalizing the result in this way is very effective.

Chapter 5

Implementation

5.1 Third party software

The implementation has used a few different third party libraries to allow the authors to focus on the problem specific to this thesis. Next these libraries will be briefly described and a motivation will be provided for why they were was chosen.

The functionality required to implement the benchmarking is fairly specific and software that deals with finance is a niche market. Two additional requirements were imposed by the authors.

- It had to be open source peer reviewed software so the source could be reviewed
- Due to a limited budget the software had to be free

Once these the requirements were clear a quick survey was done and there were two libraries that fullfilled them, QuantLib[6] and Zipline[7].

A quick review of these libraries revealed that the documentation for Zipline was superior and thus it was decided to use that. Zipline 0.6 was the most current version when the work was started so that is the version used.

The decision to use Zipline implicitly decided for us to use Python[8]. Python 3.4.0 has been used for all code in this thesis.

A Python library called matplotlib[9] has been used to generate all graphs in this thesis and version 1.3.1 has been used.

5.2 Data conversion

The data exported from NASDAQ OMX Nordic is not directly compatible with the implementation used in this thesis. Therefore the data had to be converted into a format that was supported.

This conversion is implemented in the format.sh shell script and it performs the following conversions:

- Remove the strings "price" and "total " from the CSV header

- Replace "opening" with "open" and "closing" with "close" in the CSV header
- Use the dot character instead of the comma character for decimals
- Remove all columns but the date, open, close, high and low price, the volume and the number of trades
- Add the symbol name to the data

5.3 The algorithm

MACD was chosen as a generator of signals for the thesis but it is not a complete algorithm. Therefore this had to be created by the authors of this thesis. This implementation can be found in the file macd_trading_algorithm.py that is available in the the appendix.

This implementation interprets the combination of a postitive MACD and a positive histogram as a bullish signal and the reverse as a bearish signal.

In an attempt to filter out false positives and negatives the algorithm requires two consecutive signals of either type before reacting. If the algorithm generates bearish signals for two consecutive days it will sell all holdings. If it generates bullish signals for two consecutive days the algorithm will adjust the holdings to 30% of the total portfolio value.

Chapter 6

Results

The graphs presented in this part is the result of five separated simulations run over the historical data sets of five different stocks. Every stock trading simluation is executed separetly over a ten year span begining at the 9th of April 2004 and ending at the 9th of April 2014. Each simulation had its own inital investment portfolio and they did not interact with each other in any way. This was to keep the simulation as simple as possible. In a more realistic simulation it would however be better to simulate them under one shared portfolio. Because of timelimits with implementation we did however have to run them separatly. The diagrams are normalized with all curves starting at one. A growth to two then means that it had a one hundred percent increase.

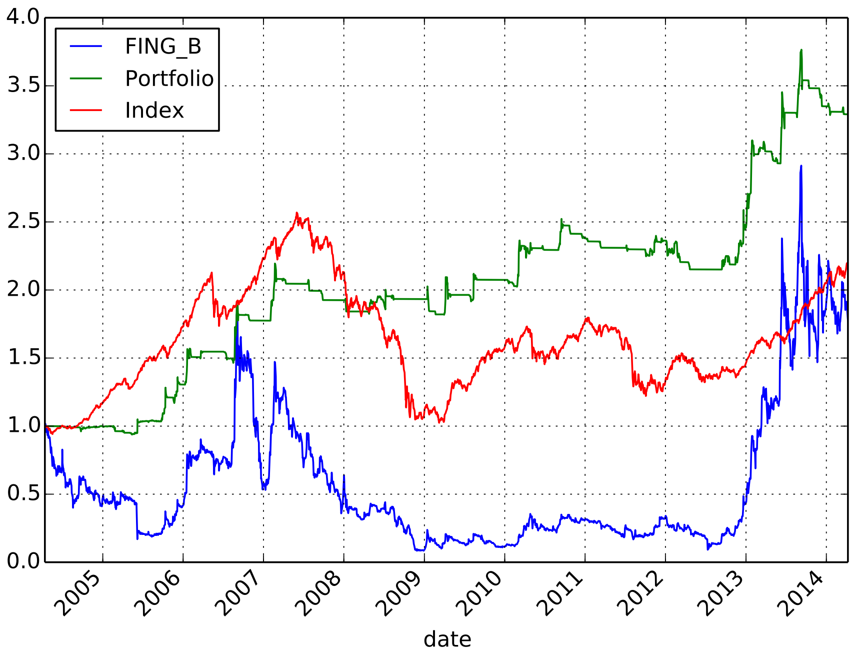

Figure 6.1. FING_B stock performance over ten years

The first stock that we benchmarked was Fingerprint cards and it showed a solid result with our algorithm giving back a profit 110% over the index. We can in the diagram see that the algorithm (green line) kept on a good level during the middle time period when the stocks price was quite low. This is because the algorithm makes a large profit at the beginning and then a big downtrend hits were the algorithm dumps its stock for a very good price. During 2007-2013 it just makes small openings and closings keeping the portfolio at a steady level. We then see another big increase in the end where the stock once again shows a powerful upswing in sell price.

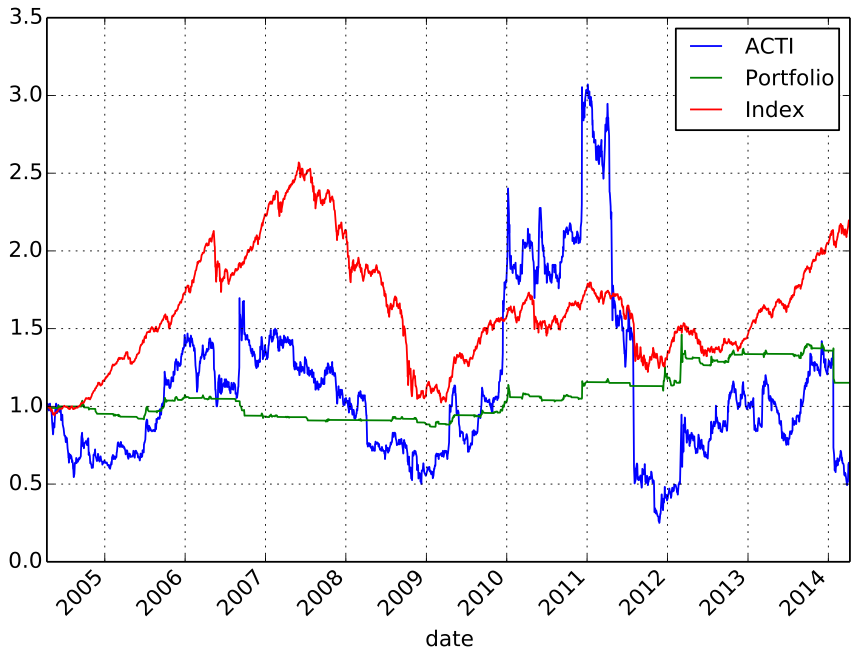

Figure 6.2. ACTI stock performance over ten years

In the second simulation the portfolio kept on a very stable level not really being affected by the big swings in the market price. In the end it did go up and it stayed over the price of the stock but well under the index.

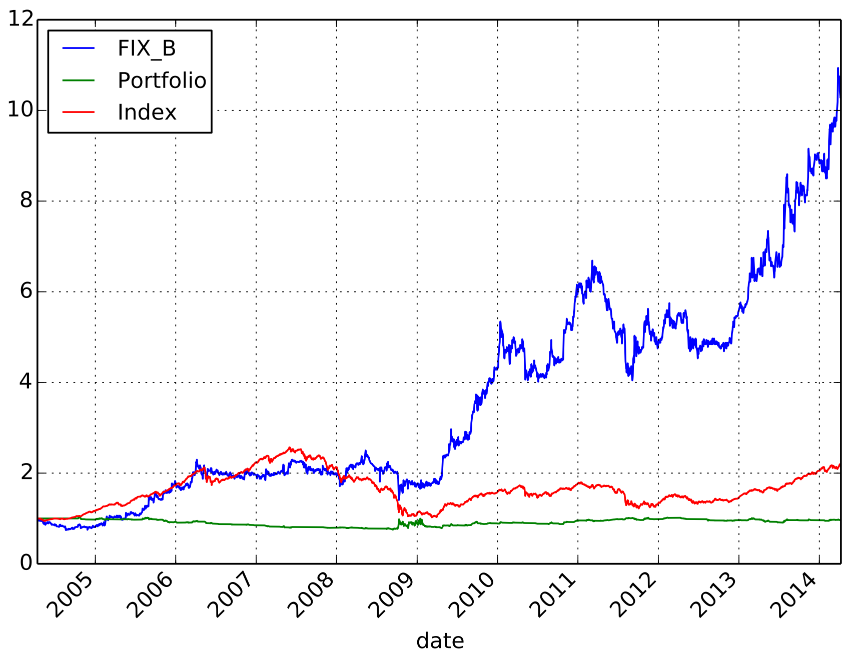

Figure 6.3. FIX_B stock performance over ten years

In this case the algorithm did not do good. It can possibly be explained by a large number of very short trends where the algorithm never were able to make any substainable profits. It can also be so simple as the MACD line not passing the signal line and therefore not giving the algorithm signals to buy.

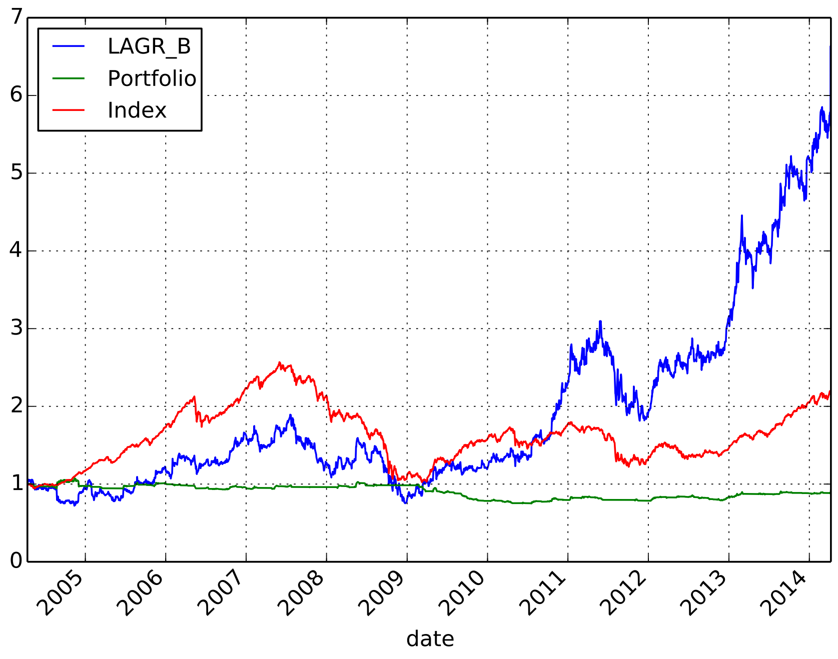

Figure 6.4. LAGR_B stock performance over ten years

We once again see the same problem where the algorithm does not receive enough signals from the MACD.

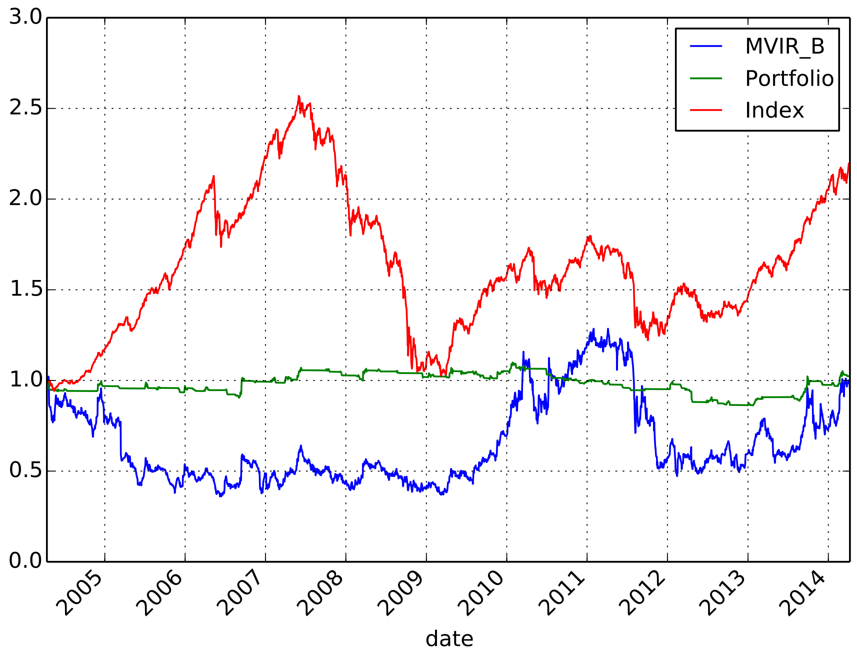

Figure 6.5. MVIR_B stock performance over ten years

Here we can see the result of a stock doing very poorly over the years. Going down from the begining and never really building any momentum. The algorithm still manages to go plus in the end. However on a ten year scale that is a negligable return.

Chapter 7

Discussion

Overall, the algorithm did not perform very well. A lot comes down to bad filtering where we lose both good sell and buy signals because we try to make sure that the trend is stable. To improve the algorithm one could implement some kind of confidence variables and extra guards signaling when to buy or sell. It is, however, difficult and it becomes risky when you try to shape your algorithm too much based on historical data as the future might look different and expected previous results may not come true at all. Improving one thing may also cost the ability to utilise on other and this is were the large complexity of economical forecasting shows itself. To really write a good algorithm we would need to study the real basics of economical signals and how to interpret them in a better way. To write such a piece of software would take a considerable amount of time and this article should be more seen as a trail-and-error research test into the universe of algorithmic trading. It is not by accident that the algorithms in use by big traders are highly guarded secrets. Writing your own algorithm shows how hard it is to forecast the market and how to respond to what is happeing. The MACD itself provides good visualization of trends but it is hard to predict how long one will continue. It is possible to automate large part of the stock trading but as we have seen and as it is in real life trading aswell it is very important with human supervision. We have implemented such a small set of signals and it is in the end a very basic trading algorithm. Too make it more sophisticated you need more input and ways to control that trends are what they look like at first. Humans can do this by studying social channels and other sources and use the algorithm more as a tool to help in the trading process. This is of course possible to extend in to fully automatic trading using algorithms but to do this we would have needed to create a alot more extencive tools to gather social data or other signals. The more information there is to proccess and use as signals, the more accurate a trader could hopefully be.

Appendices

Appendix A

Source code

A.1 common.py

```
import pandas
import pytz

from matplotlib.dates import date2num
from pandas.stats.moments import ewma
from dateutil.parser import parse

def load_eod_data(file):
    eod_data = pandas.read_csv(file,
                               delimiter=';',
                               parse_dates=True,
                               date_parser=lambda x: parse(x).replace(tzinfo=pytz.utc),
                               index_col='date')

    if len(eod_data) == 0:
        raise SystemExit

    eod_data['dt'] = eod_data.index.map(lambda x: x.replace(tzinfo=pytz.utc))

    eod_data['date'] = eod_data.index.map(date2num)

    return eod_data

def compute_macd(data):
    data['ewma5'] = ewma(data.close, span=5)
    data['ewma35'] = ewma(data.close, span=35)

    data['macd'] = data.ewma5 - data.ewma35
    data['macd_signal'] = ewma(data.macd, span=5)

    data['histogram'] = data.macd - data.macd_signal
```

A.2 data_frame_source.py

```
import numpy
import pandas
import logbook

from zipline.gens.utils import hash_args

from zipline.sources.data_source import DataSource

log = logbook.Logger('DataFrameSource')

class DataFrameSource(DataSource):
    def __init__(self, data, **kwargs):
        assert isinstance(data.index, pandas.tseries.index.DatetimeIndex)

        self.data = data
        self._raw_data = None

        # Unpack config dictionary with default values.
        self.start = kwargs.get('start', data.index[-1])
        self.end = kwargs.get('end', data.index[0])

        # Hash_value for downstream sorting.
        self.arg_string = hash_args(data, **kwargs)

    @property
    def mapping(self):
        return {
            'dt': (lambda x: x, 'dt'),
            'sid': (lambda x: x, 'sid'),
            'open_price': (float, 'open'),
            'close_price': (float, 'close'),
            'price': (float, 'close'),
            'high': (float, 'high'),
            'low': (float, 'low'),
            'volume': (float, 'volume'),

            'macd': (float, 'macd'),
                'macd_signal': (float, 'macd_signal')
            }

    @property
    def instance_hash(self):
        return self.arg_string

    def raw_data_gen(self):
        for dt, series in self.data.iterrows():
            if numpy.isnan(series.volume):
                log.warn("Cleaning event at %s from nan volume" % dt)
                series.volume = 0

            yield series

    @property
    def raw_data(self):
        if not self._raw_data:
            self._raw_data = self.raw_data_gen()
        return self._raw_data
```

A.3 macd_trading_algorithm.py

```
import logbook

from numpy import nan, isnan
from zipline.algorithm import TradingAlgorithm
from zipline.finance.commission import PerTrade
from zipline.finance.slippage import VolumeShareSlippage

log = logbook.Logger('MACDTradingAlgorithm')

class MACDTradingAlgorithm(TradingAlgorithm):
    def __init__(self, *args, **kwargs):
        self.sid = kwargs.pop('sid')
        super().__init__(*args, **kwargs)

        self.bullish = 0
        self.bearish = 0

    def initialize(self):
        self.set_commission(PerTrade(cost=99.0))
        self.set_slippage(VolumeShareSlippage(volume_limit=0.25, price_impact=0.1))

    def handle_data(self, data):
        for symbol, stock in data.iteritems():
            position = self.portfolio.positions[symbol]

            histogram = stock.macd - stock.macd_signal

            if histogram > 0 and stock.macd > 0:
                self.signal_bullish()

                if self.bullish > 1:
                    log.info("Longing on %s at %f close price (cash: %d, portfolio_valu
                            (symbol, stock.close_price, self.portfolio.cash, self.portf
                    self.order_target_percent(self.sid, 0.3)
            else:
                self.signal_bearish()

                if self.bearish > 1 and position.amount > 0:
                    log.info("Shorting %d %s at %f close price" % (position.amount, sym
                    self.order(symbol, -position.amount)

    def signal_bearish(self):
        self.bearish += 1
        self.bullish = 0

    def signal_bullish(self):
        self.bullish +=1
        self.bearish = 0
```

A.4 trading.py

```
#!/usr/bin/env python3

import sys
import optparse
import pytz
import pandas

import matplotlib.pyplot as plot
import zipline.utils.factory as factory

from matplotlib.dates import AutoDateLocator, AutoDateFormatter

```

```python
from data_frame_source import DataFrameSource
from macd_trading_algorithm import MACDTradingAlgorithm
from common import load_eod_data, compute_macd

def parse_args():
    parser = optparse.OptionParser(usage="usage: %prog [options] SYMBOL [FILE]")
    parser.add_option("-s", "--save", action="store", dest="save")
    parser.add_option("-p", "--plot-price", action="store_true", dest="plot_price")
    parser.add_option("-m", "--plot-macd", action="store_true", dest="plot_macd")

    (options, args) = parser.parse_args()

    if len(args) < 1:
        parser.error("You need to provide the symbol as the first argument")

    symbol = args[0]

    file = sys.stdin
    if len(args) > 1:
        file = args[1]

    num_subplots = 1
    if options.plot_price:
        num_subplots += 1
    if options.plot_macd:
        num_subplots += 1

    return (file, symbol, num_subplots, options.plot_price, options.plot_macd, options.

def draw_plot(num_subplots):
    fig, axes = plot.subplots(nrows=num_subplots, sharex=True)

    major_locator = AutoDateLocator()
    minor_locator = AutoDateLocator()

    if num_subplots == 1:
        axes = [axes]

    axes[0].xaxis.set_major_locator(major_locator)
    axes[0].xaxis.set_minor_locator(minor_locator)

    date_formatter = AutoDateFormatter(major_locator)
    axes[0].xaxis.set_major_formatter(date_formatter)

    axes[0].xaxis_date()
    axes[0].autoscale_view()

    return fig, axes

def draw_price(symbol, data, axis):
    axis.plot(data.date, data.close, 'g--', label=symbol)
    axis.plot(data.date, data.ewma5, color='red', lw=1, label='EMA5')
    axis.plot(data.date, data.ewma35, color='blue', lw=1, label='EMA35')
    axis.legend(loc='best', fontsize='medium')

def draw_macd(data, axis):
    axis.plot(data.date, data.macd, color='red', lw=1, label='MACD')
    axis.plot(data.date, data.macd_signal, color='green', lw=1, label='MACD Signal')
    axis.fill_between(data.date, data.histogram)
    axis.legend(loc='best', fontsize='medium')

def draw_benchmark(symbol, data, results, idx_data, axis):
    normalize(data.close).plot(ax=axis, label=symbol)
    normalize(results.portfolio_value).plot(ax=axis, label='Portfolio')
    normalize(idx_data.close_price).plot(ax=axis, label='Index')
    axis.legend(loc='best', fontsize='medium')


def normalize(data):
    return data / data[0]


if __name__ == "__main__":
    (file, symbol, num_subplots, plot_price, plot_macd, save) = parse_args()

    # set up the custom data source and the trading algorithm
    eod_data = load_eod_data(file)
    compute_macd(eod_data)

    # set up the the simulation parameters for the backtesting
    sim_params = factory.create_simulation_parameters(start=eod_data.index[1],
                                                      end=eod_data.index[-1])

    source = DataFrameSource(eod_data)
    algo = MACDTradingAlgorithm(sid=symbol, sim_params=sim_params)

    # run the algo
    results = algo.run(source)
```

```python

    idx_eod_data = load_eod_data("./SE0001776618-2004-04-09-2014-04-09.csv")

    fig, axes = draw_plot(num_subplots)

    plot_idx = 0
    if plot_price:
        draw_price(symbol, eod_data, axes[plot_idx])
        plot_idx += 1
    if plot_macd:
        draw_macd(eod_data, axes[plot_idx])
        plot_idx += 1

    draw_benchmark(symbol, eod_data, results, idx_eod_data, axes[plot_idx])
    plot.setp(plot.gca().get_xticklabels(), rotation=45, horizontalalignment='right')

    if save:
        plot.savefig(save, format='pdf', transparent=True)
    else:
        plot.show()
```

A.5 format.sh

```bash
#!/usr/bin/env bash

set -eu

rm -rf data
cp -r data_orig data

# Don't expand to *.csv when there are no matching files
shopt -s nullglob

CSV_FILES="./data/*.csv"

for file in $CSV_FILES
do
  echo "Converting $file"

  sed -i '1 s/\(.*\)/\L\1/' $file
  sed -i '1 s/ price//g' $file
  sed -i '1 s/total //g' $file

  sed -i '1 s/opening/open/' $file
  sed -i '1 s/closing/close/' $file

  # Use dot for decimals
  sed -i 's/,/./g' $file

  # Remove the unwanted columns
  cut -d ';' -f1,4-7,9,11 $file > $file.back
  mv $file.back $file

  symbol=$(basename $file | cut -d "-" -f1)
  header=$(head -n1 $file)

  # Sort the file based on the first column
  # (date), removes the header
  tail -n+2 $file | sort > $file.back
  mv $file.back $file

  # Put the header back
  sed -i "s/^/$symbol\;/" $file
  sed -i "1isid;$header" $file
done
```

Bibliography

[1] N. Ulku and E. Prodan, “Drivers of technical trend-following rules”, *International Review of Financial Analysis*, vol. 30, pp. 214–229, 2013.

[2] M. A. Goldstei and P. K. andFrank C. Graves, “Computerized and high-frequenct trading”, *Financial Review*, vol. 49, no. 2, pp. 177–202, 2014.

[3] C. D. K. II and J. R. Dahlquist, second edition. Pearson Education, 2006.

[4] G. Apple, first edition. Financial Times Prentice Hall, 1999, p. 166.

[5] NASDAQ OMX Group, Inc. (Apr. 2014). Historiska kurser - aktier - NASDAQ OMX NORDIC, [Online]. Available: `http://www.nasdaqomxnordic.com/aktier/historiskakurser` (visited on 04/29/2014).

[6] Ferdinando Ametrano. (Apr. 2014). QuantLib: a free/open-source library for quantitative finance, [Online]. Available: `http://quantlib.org/` (visited on 04/29/2014).

[7] Quantopian, Inc. (Apr. 2014). quantopian/zipline, [Online]. Available: `https://github.com/quantopian/zipline` (visited on 04/29/2014).

[8] Python Software Foundation. (Apr. 2014). Welcome to Python.org, [Online]. Available: `https://www.python.org/` (visited on 04/29/2014).

[9] J. D. Hunter, “Matplotlib: a 2d graphics environment”, *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.