MULTIFRACTAL MODEL: FORECASTING STOCK RETURNS IN THE UKRAINIAN STOCK MARKET

BY

Andrii Riabushenko

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

Master of Arts in Economics

National University "Kyiv-Mohyla academy"
Master's program in Economics

2008

Approved by

Mr. Mr. Volodymyr Sidenko (Head of the State Examination Committee)

Program Authorized
to offer the degree Master's Program in Economics, NaUKMA

Date

National University “Kyiv-Mohyla academy”

Abstract

MULTIFRACTAL MODEL:
FORECASTING STOCK RETURNS IN
THE UKRAINIAN STOCK MARKET

by Andrii Riabushenko

Head of the State Examination Committee:
Mr. Volodymr Sidenko, Senior Economist
Institute of Economy and Forecasting,
National Academy of Sciences of Ukraine

This study investigates stock prices volatility and asset returns in transition economies, specifically Ukraine. Transition countries have volatile stock markets compared to developed countries. This makes classical GARCH models perform poor forecast in marketos of transition countries. This study proposes a new model - multifractal Markov-switching model (MSM) to address limitations of GARCH models. The multifractal model uses fractal properties found in the stock returns : (i) long memory, (ii) lags decay hyperbolically, slower than exponentially, (iii) not normal fat-tailed distribution, (iii) different degree of long memory. These properties significantly increase the forecasting accuracy of MSM model. Three criteria where used to evaluate and compare the forecasting power of MSM and GARCH models: relative MAE, relative MSE and portfolio annual return in Ukrainian PFTS stock market in 2007 balanced with the help of the forecast of expected return and risk. All three criteria indicate that MSM model significantly outperforms the GARCH model in the stock market of the transition countries. The annual portfolio return balanced with the help of MSM forecast outperformed the market return by 27.35%, while annual return of the portfolio based on the GARCH forecast showed 6% less then market.

Table of contents

List of figures

List of tables

Glossary

ARCH — Autoregressive conditional heteroscedasticity . . . 13

GARCH — Generalized autoregressive conditional heteroscedasticity . . . 13

Multifractal Markov-switching model (MSM) — stationary time series model that models innovations as set of fractionally random walks with regime switching among them . . . 13

Fractal — a geometric shape that exhibits two properties: invariance under displacement and invariance under scale . . . 15

Fractal dimension — generalization of typological dimension that measures the number of objects that fits the scaled bounds . . . 15

Hurst exponent (H) — the measure of fractal dimension of time series named in honor of Harold Edwin Hurst. The usefulness of Hurst exponent is that it can be easily estimated from data. . . . 16

Brownian motion — is a continuous-time stochastic processes such that all increments $\triangle B(t)$ are independent, equally probable, and random . . . 16

Long memory process — is a path dependent process with all innovations correlated, the very first innovation influence all future innovation . . . 18

Fractional Brownian motion(fBM) — generalization of Brownian motion that allows to model long-memory and path dependence in time-series process . . . 18

FIGARCH — fractionally integrated GARCH . . . 19

Introduction

This study investigates stock prices volatility and stock returns in transition economies, specifically Ukraine. Transition economies have highly volatile stock markets compared to developed countries as demonstrated in section 3.3. This makes many asset returns models not applicable. For example, multi-factor models are a dominant framework for forecasting asset returns in US. However, multi-factor models perform very poorly in transition countries because performance of transition countries stock returns are not tied to macroeconomic indicators such as inflation and GDP growth rate [Perevozchikov 2007].

Diverse econometric models were developed to address volatility of emerging financial markets. However, to the best of my knowledge, no investment company in Ukraine has used regression methods to forecast asset returns because of poor performance of conventional models such as ARCH and GARCH in emerging markets, as demonstrated in section 5.1. While, in developed countries the GARCH models are widely used by the industry of the short-term investment. I hope this will subject to change with this study, and regression methods will be used by Ukrainian investment companies since limitations of GARCH models in transition economies can be addressed by Multifractal Markov-switching model (MSM).

Multifractal Markov-switching model can model almost all properties found in financial data of emerging markets, specifically: long memory, fat-tailed distribution and others. Therefore, multifractal Markov-switching model has a chance to become a dominant framework in forecasting asset returns in Ukraine.

The study starts with theoretical background to introduce the concept of fractals to the reader and their applications in time-series analysis. The next chapter is devoted to short literature review of latest advances and applications of GARCH and fractal models in stock returns forecasting. The third chapter describes the data on Ukrainian PFTS stock returns that was used in this study. Also the proper tests for long-memory, non normality are conducted in order to check whether fractal model are applicable in Ukraine. The next chapter briefly describes the multifractal Markov switching model and the ways to estimate it. The fifth chapter fully devoted to evaluation of forecasting accuracy of multifractal model based on the relative MSE and MAE criteria on PFTS stock exchange. The section 5.2 worth a separate notice. It offers another approach to evaluate forecasting power - the annual portfolio return earned in PFTS stock exchange in 2007. The portfolio is optimized daily based on the MSM forecast of expected returns and risk. Transaction costs and bid-ask spread are taken into account to make conditions as close to real stock exchange conditions as possible.

Chapter 1
Theoretical background

1.1 The concept of fractals

Fractal doesn't have a strict mathematical definition. Mandelbrot, in his early studies, defined fractal as a geometric shape that exhibits two properties: (i) invariance under displacement, and (ii) invariance under scale [Mandelbrot 1977]. The object is invariant under displacement, if any part of the object looks similar to all other parts. The object is invariant under scale if a magnified object looks similar to the object itself. Classical mathematics has problems with defining such structures as fractals, and therefore, the fractal is usually defined through its properties.

Geometric object F is fractal if it satisfies four properties:

— F has detail at every scale;

— F is exactly or statistically self-similar;

— F has a recursive definition;

— Fractal dimension of F is different from topological dimension of F.

The last property requires a definition of fractal dimension. Fractal dimension is the generalization of typological dimension. Fractal dimension measures the number of objects that fits the scaled bounds.

$$\text{Fractal dimension} = \lim_{e \to 0} \frac{\log N(e)}{\log \frac{1}{e}}$$

Where N is the number of geometric objects visible at given scale and $\frac{1}{\varepsilon}-$ is the scale magnification factor. Fractal dimension measures how much space an object occupies. Classical geometric figures like:

- points have topological dimension 0
- Lines and curves have typological dimension 1
- Square, rhomb and etc. have typological dimension 2
- Cube, cylinder, cone and etc. have typological dimension 3

Their fractal dimension is exactly equal to typological dimension. That is why a straight line is not a fractal because the last property of fractal requires inequality between the typological and fractal dimensions.

While the fractal dimension is a good measure to compare fractals with classical Euclidean shapes, it is often more suitable to use another fractal measure - the Hurst exponent (H), because the Hurst exponent can be easily estimated on the basis of empirical data. The fractal dimension, and the Hurst exponent are in one-to-one relationship which can be expressed as follows:

$$D = 2 - H$$

where D - is fractal dimension and H is Hurst exponent.

The fractals can be used to model geometric shapes that are interesting to statisticians. The Brownian motion is an example of such shape.

1.2 Brownian motion and random walk

The Brownian motion is a function $B(t)$ defined for equally-spaced time steps $\triangle t$, such that all increments $\triangle B(t)$ are independent, equally probable, and random [Mandelbrot 1977]. The Brownian motion exhibits many properties that make it

useful for time series analysis such as: Brownian motion is a Markov process [Kaye 1989] since its future state is determined only by current state and is independent from the past; hence, the Brownian motion is a stationary process.

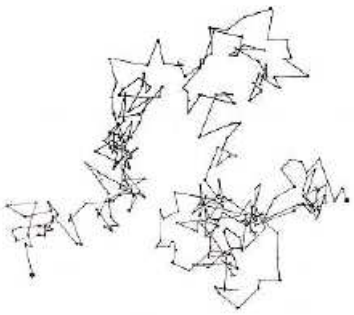

Figure 1.1. Example of Brownian motion [Mandelbrot 1977]

One-dimensional random walk is a discrete time process that is described by equation (1.1):

$$X(t+\triangle t) = X(t) + \varepsilon \tag{1.1}$$

Most of econometric time series estimators model innovations as random walks. Random walk is closely related to the Brownian motion. Random-walk with small steps is an approximation of Brownian motion. Convergence of random-walk to the Brownian motion is proved by a central limit theorem. Position of a "walker" after large number of steps is normally distributed with mean zero and variance $\sigma^2 = n\varepsilon^2$, where n number of steps since start of random walk, ε - size of the step. This convergence is heavily used by time series estimators like AR, ARIMA, GARCH.

The Brownian motion is a pure true fractal since it is statistically self-similar, and its fractal dimension is equal to $\frac{3}{2}$ [Falconer 1985]. On the other hand, random walk is not a pure fractal but is rather its discrete version.

1.3 Fractional Brownian motion and fractional random walk

Mandelbrot used his fractal self-similarity concept to relax the assumption of independent position increments $\triangle B(t)$ in the Brownian motion and generalise the Brownian motion for path dependent processes. Such process is called the fractional Brownian motion (fBm). The increments $\triangle B(t)$ of the process are correlated in a such way that:

$$E(B^H(t)B^H(s)) = \frac{1}{2}(t^{2H} + s^{2H} - |t-s|^{2H})$$

- If $H = \frac{1}{2}$ then fractional Brownian motion becomes a regular Brownian motion, as $E(B^H(t)B^H(S)) = \frac{1}{2}(|t| + |s| - |t-s|) = \begin{cases} t, \text{if } t \geqslant s \\ s, \text{if } t < s \end{cases}$
- If $H > \frac{1}{2}$, the increments of the process are positively correlated. Such process is called persistent process and it has a long memory, as $\sum_{n=1}^{\infty} E(B^H(1)(B^H(n+1) - B^H(n)) = \infty$.
- If $H < \frac{1}{2}$, the increments of the process are negatively correlated. Such process is called anti-persistent.

Fractional Brownian motion is still a pure fractal because it is statistically self-similar $\left(B^H(a\,t) \sim |a|^H B^H(t)\right)$ and its Hurst exponent is exactly equal to H, and its fractal dimension is equal to $D = 2 - H$.

As in the case of a regular Brownian motion, there is a semi-random walk process that is related to the fractional Brownian motion as a random walk. This semi-random walk process is a generalization of random walk with dependent movements, and it is called fractional or fractal random walk. Modelling the financial time series as fractional random walk allows to capture long memory that is found in stock returns.

Someone could argue that central limit theorem predicts that walker position after large number of steps will converge to normal distribution even if walker follows fractional random walk and it would be still possible to use estimators that a

based on Brownian motion. But this is a rare case when central limit theorem doesn't apply. The definition of central limit theorem states that the sum of n random variables with expectation E and variance σ^2 tends to normal distribution with mean E and variance $\frac{\sigma^2}{n}$ as n approaches infinity. But the distribution of the position of the fractional random walk has infinite variance, as it will be shown later on in section 3.3. The central limit theorem doesn't hold for distributions that have infinite second moment.

1.4 Financial time series as fractional random walks

As it is shown in the section 3.3, stock returns in Ukraine stock market follow fractional random walk, and therefore, satisfy all properties of fractals. That is why asset returns in Ukraine can be treated as fractals. In such a case, GARCH model with fractional random walk can be applied. It is called FIGARCH model.

Moreover, fractal dimension of asset returns varies over time. In order to fix this time rescale inconsistency, the Markov chain regime switching model will be applied. This approach allows us to define several regimes, in our case several fractal dimensions, and switch across regimes over time. As a result, joint model of FIGACH and Markov chain regime switching provide us with multifractal model. Multifractal is generalization of fractal, which does not require fractal dimension to be constant. Moreover, fractal dimension of multifractal is defined as function of time.

1.5 A Critique of the Efficient Market Hypothesis

In 2000, hundreds of IT companies went bankrupt, and the NASDAQ index lost 50% over a short period of time. Fortunes were made and lost during such rapid changes, markets' volatility speeded up and became hardly predictable. Classical

efficient market hypothesis (EMH), which is a common practice, can not explain these processes. According to EMH, probability of such crisis like January 2000 is not higher than 10^{-12}. However, spikes on financial markets occur on regular basis.

EMH claims that as soon as a new stock appears in the market, its price equals the fair value and follows a random walk. Do financial prices follow the prediction of EMH theory? Of course, they never do, and the reason is obvious. The EMH is based on non realistic assumptions, such as:

— Capital markets are absolutely efficient;

— All investors are homogeneous: All investors share the same utility function and discount rate;

— Stock prices follow a random walk.

If these assumptions hold then no trade among investors would occur, because all investors are identical. All of them want to buy the same stock at the same time, but no seller for the stock and vice-versa. However, do we observe this kind of behaviour of the market? Definitely not, daily trade on stock exchange reaches billion of dollars every day.

EMH says nothing about the liquidity. EMH assumes that stock prices are always fair and provide enough liquidity, but most of financial crashes and stampedes happened during periods of low liquidity and imbalanced trading volume. Hence, EMH can't explain the reasons of stock markets failures.

1.6 Fractal market hypothesis

The fractal market hypothesis(FMH) tries to address limitations of EMH. The idea of fractal market hypothesis was first stated in [Peters 1991]. FMH emphasizes the importance of investment time horizon and liquidity. Also, FHM, unlike

EMH, does not impose assumptions on the distribution of price changes. Investors require liquidity from markets. Liquidity ensures that investors can trade effectively with one another. Different investors with different time horizons can trade and react to the signals provided by stock exchange. Reaction of an investor with 15-minute time horizon to the fall in stock price would be to sell the stock not to lose more since prices are unlikely to go up in 15 minutes, while institutional investor, like pension fund, with long time horizon would react differently. Its optimal decision would be to buy the stock while price is lowered. The markets are stable, as long as markets provide enough liquidity - negative short-term fluctuations are balanced by long-term investors, positive short-term fluctuations are favoured by short-term investors.

Now let us assume that an external shock such as political crisis happened. The long-term investors are uncertain about the future, and therefore, they leave the market or become short-term investors causing a decrease in liquidity. The market looses its stability: when negative short-term fluctuations take place, short-term investors start to sell stocks, and no one is willing to equilibrate the market. Prices start to fall rapidly causing financial crisis. Hence, what is observed on the market is the change of average time horizon of the market. The fractals are self-similar objects under any scale of time, therefore fractals capable to capture this shifts of time horizon in a natural way. Therefore, the fractals can be used to forecast big events on the stock market such as financial crisis.

Chapter 2
Literature review

ARCH [Engle 1982] and GARCH [Bollerslev 1986] were the first volatility models that made forecasting of assets returns possible. This was an enormous success in financial econometrics. GARCH models have become a dominant representation of financial prices for quite a long time. GARCH is defined as:

$$p_t = p_{t-1} + \varepsilon_t, \varepsilon_t = u_t\sqrt{h_t} \tag{2.1}$$

u_t are independently identically distributed, $u_t \sim N(0, \sigma^2)$

$$h_t = \beta_0 + \sum_{i=1}^{n} \beta_i \varepsilon_{t-i}^2 + \sum_{j=1}^{k} \beta_i h_{t-j}$$

The GARCH model has enormous number of extensions, among which most prominent are GARH-M, EGARCH, TGARCH. The GARCH-M model introduced not only conditional variance, but also conditional mean (2.2).

$$p_t = p_{t-1} + \sqrt{\sigma_t} + \varepsilon_t \tag{2.2}$$

The exponential GARCH (EGARCH) model allowed for non-linear specification of conditional variance

$$\log \sigma_t^2 = w_t + \sum_{k=1}^{\infty} \beta_k g(Z_{t-k})$$

Finally, the Threshold GARCH (TGARCH) models the asymmetry in GARCH process:

$$\sigma_t = K + \delta\sigma_{t-1} + \beta_1^+ \varepsilon_{t-1}^+ + \beta_1^- \varepsilon_{t-1}^-$$

However, in the late 1980's, asset returns were found to exhibit fractal properties, such as:

– long memory, the infinite lag length;

– lags decay hyperbolically, slower than exponentially;

– returns to assets follow not normal fat-tailed distribution;

– different degree of long memory, which known as time rescaling inconsistency.

Intuitively, the long memory process is when all innovations in time series are correlated, no matter how large the sample size is. Long memory was first reported by Taylor [Taylor 1986], and now it is intensively used in financial time series for both stock prices and exchange rates.

Lags decay hyperbolically, and long memory implies that lags decrease slower than exponentially. Specifically, the financial prices decay hyperbolically causing difficulties during estimation procedure.

Time rescaling inconsistency is the different level of conditional heteroscedasticity and long memory in different time periods. Therefore, fractional random walk is not enough to model financial time series since it provides a single level of long memory and conditional heteroscedasticity. In order to overcome this limitation, multifractal model was developed. A complete description of financial data properties can be found in [Peters 1994].

Eventually the list of GARCH limitations became too long, and it became clear that some better estimators should be developed for stock returns and volatility. The first such estimator that could correspond to fractal properties was FIGARCH, first introduced in [Baillie *et al.* 1996]. The FIGARH model is based on ARFIMA(p, q, d) process just like GARCH is based on ARIMA(p, q, d) process. The improvement of ARFIMA(p, q, d) is that parameter d (difference) is allowed be any real number instead of integer. If d is an integer value the ARIMA and ARFIMA processes are identical. This FIGARCH fractal estimator significantly improved GARCH. FIGARCH models address the issues of long-memory

and not normal fat-tailed distribution, but couldn't generate different levels of long-memory for different powers of returns - the property that has been found in almost all financial data - thus, they couldn't address the problem of time rescaling inconsistency. Nevertheless, predictive power of FIGARCH model was significantly higher than that of GARCH. Therefore, very quickly FIGARCH have become popular in finance [Mandelbrot *et al.* 1996].

The idea of generalizing fractal estimator to multifractal estimator appeared after the adoption of FIGARCH. However, this idea faced several difficulties. In particular, non-stationarity and the combinatorial nature of the multifractal model have been overcome by the introduction of an iterative Markov-switching multifractal model (MSM) introduced in [Calvet and Fisher 2001]. Afterwards, MSM model was significantly improved by regime switching. Regime switching serves three purposes: It captures (i) low frequency variations, (ii) intermediate frequency for smoothing autoregressive processes and finally, (iii) high frequency switches generate substantial outliers [Calvet and Fisher 2004]. The multifractal model resolved the issue of time rescaling inconsistency since regime switching allowed for different level of long-memory and conditional heteroscedasticity for different powers of returns.

Finally, Thomas Lux (2006) has developed GMM estimation procedure for multifractal Markov-switching models to overcome the limitations of ML estimation procedure and reduce its computational complexity, which required volatility components to follow discrete distribution [Lux 2006].

The first application of MSM model based on GMM estimation was carried out on Japan stock market [Lux and Kaizoji 2006]. The application showed that MSM model is as effective as FIGARCH model in the short term forecasts (single day), but MSM model performs much better in the long term forecasts such as 50-100 days [Lux and Kaizoji 2006]. Mean square error of MSM estimator increases very slowly over time compared to GARCH and FIGARCH estimators.

MSM models have not been applied in the transition countries stock markets yet. This study fills this gap. The transition countries have much more volatile markets and regime switching in market occurs more frequently than in case developed countries as it is showed in section 3.3. Hence, MSM model has much more to offer stocks markets of transition countries than for developed countries.

Chapter 3
Data

3.1 Data sources

Ukraine has several stock exchanges. The PFTS stock exchange is largest among Ukrainian marketplaces for stocks. The PFTS index has earned the status of primary indicator of Ukrainian stock exchange. In 2007 PFTS index basket contained 18 stocks, all of them are listed in the table 3.1. Stocks of PFTS Index are regarded in this study, all forecast methods are checked on this 18 stocks.

The data for stocks prices of PFTS Index were provided by the asset management company Kinto. Kinto has archive of all PFTS quotes and transactions starting from the very first day of PFTS stock exchange. This data is freely available on the their website (http://kinto.com/eng/research/marketupdate/quotes.html). The collected data covers the period starting from 1997-10-10 till 2008-02-11.

The data for NYSE Index for same period as for PFTS Index was collected at Yahoo! Finance (http://finance.yahoo.com).

	Ticker	Company name
1	DNEN	Dniprenergo
2	MMKI	Ilyich Iron and Steel Works
3	ZAEN	Zakhidenergo
4	CEEN	Centerenergo
5	UNAF	Ukrnafta
6	UTEL	Ukrtelecom
7	NITR	INTERPIPE Nyzhnodniprovsky Tube-Rolling Plant
8	MSICH	Motor Sich
9	BAVL	Raiffeisen Bank Aval
10	AZST	Azovstal Iron and Steel Works
11	USCB	Ukrsotsbank
12	STIR	Stirol
13	DOEN	Donbasenergo
14	KIEN	Kyivenergo
15	LPTL	Luganskteplovoz
16	MZVM	Mariupol Heavy Machine Building Plant
17	PGOK	Poltava Ore Mining and Processing Plant
18	SMASH	Sumy Frunze Machine Building Plant

Table 3.1. PFTS Basket in 2007 (18 stocks)

3.2 Data manipulations

All companies in the PFTS basket have a long quoting history starting at 1997 thus the time series are long enough to capture long memory. However, from 1997 till 2004 PFTS stock was rather young and illiquid in early days of existence. During that period zero trade days occurred, causing missing observations. Missing observations were interpolated by cubic spline. Also, during that period price jumps from close to zero price occurred causing enormous returns, outliers. Such outliers caused problems during GARCH Maximum likelihood estimation, therefore all returns that were more than 100% per day were smoothed out by cubic spline interpolation. Luckily, there were not many outliers only 4 stocks contained from 2 to 5 outliers and none of them occurred in 2007, which the year of interest in this study.

3.3 Data description

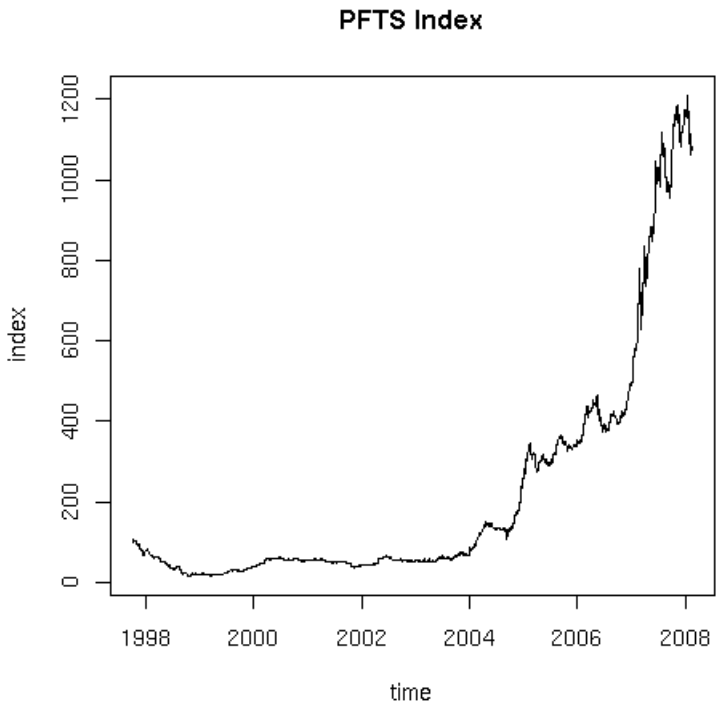

Figure 3.1. PFTS Index (2480 observations)

The PFTS index is illustrated on figure 3.1. We can see PFTS Index increased significantly during period from 2006 to 2008, especially in 2007. The PFTS Index jump increased from 497.66 to 1174.02 points in one year, bringing 135% return. The simulation of 2007 year will be a tough challenge for GARCH and MSM forecasting methods, as they better perform in equilibrium or near equilibrium conditions.

The PFTS daily returns are shown on figure 3.2A. It is clearly seen from the figure that PFTS returns daily returns are more volatile than daily returns of a developed country stock markets such as NYSE. The NYSE daily returns are presented on figure 3.2B for comparison.

The probability density function of PFTS returns estimated by KDE (Kernel density estimator) is illustrated on figure 3.3 accompanied by a normal distribution with the same mean and variance. The figure 3.3 demonstrated that PFTS daily returns are not distributed normally, they follow fat-tailed distribution. More formal tests for normality are provided in the table 3.2, four tests for normality are conducted:

— Kolmogorov–Smirnov test

— Shapiro-Wilk's test

— Jarque–Bera test

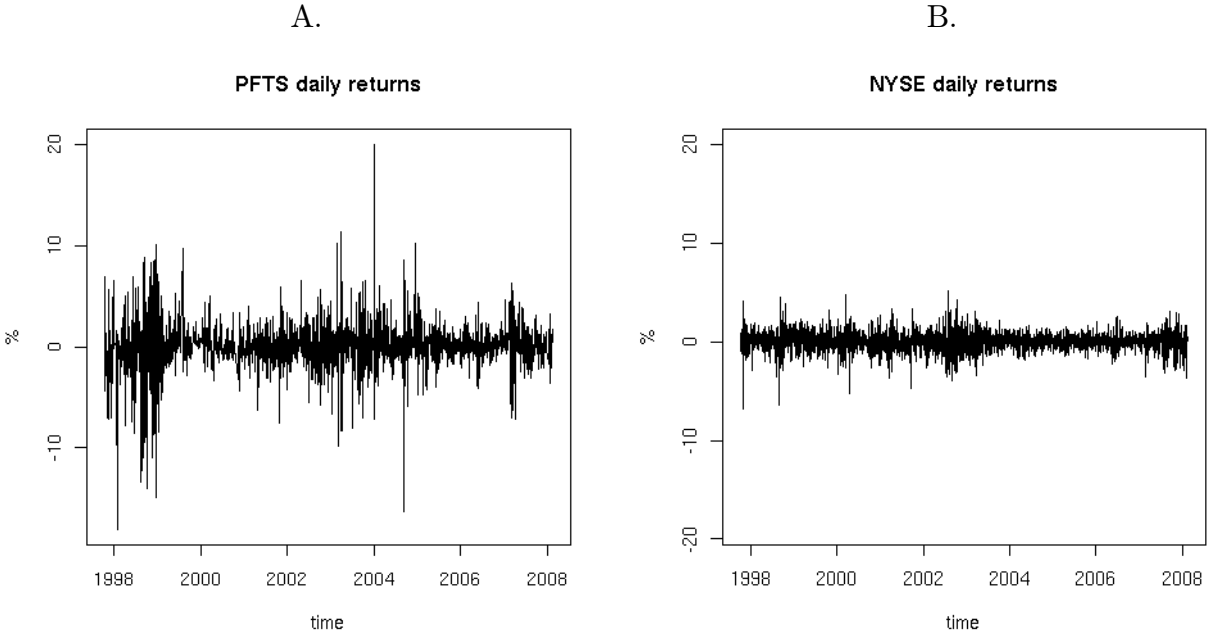

Figure 3.2. A. PTFS daily returns (2479 observations)
B. NYSE Index daily returns (2605 observations)

— D'Argostino skewness, kurtosis and omnibus normality tests

All four tests claim that we can reject the null hypothesis that returns are normally distributed with probability of very close one. The $p-$value of $2.2\text{e}-16$ is maximum precision threshold in the R package, therefore they are even lower than $2.2\text{e}-16$. Consequently, we can infer that PFTS Index returns are not normally distributed, but follow a fat-tailed distribution.

Test	Statistic	p-value
Kolmogorov-Smirnov	0.4662	$< 2.2\text{e}-16$
Shapiro-Wilk's	0.8651	$< 2.2\text{e}-16$
Jarque–Bera	12819	$< 2.2\text{e}-16$
D'Agostino	Omnibus=574.8643	$< 2.2\text{e}-16$
	Skewness=-10.3011	$< 2.2\text{e}-16$
	Kurtosis=21.6507	$< 2.2\text{e}-16$

Table 3.2. Normality tests of PFTS index returns

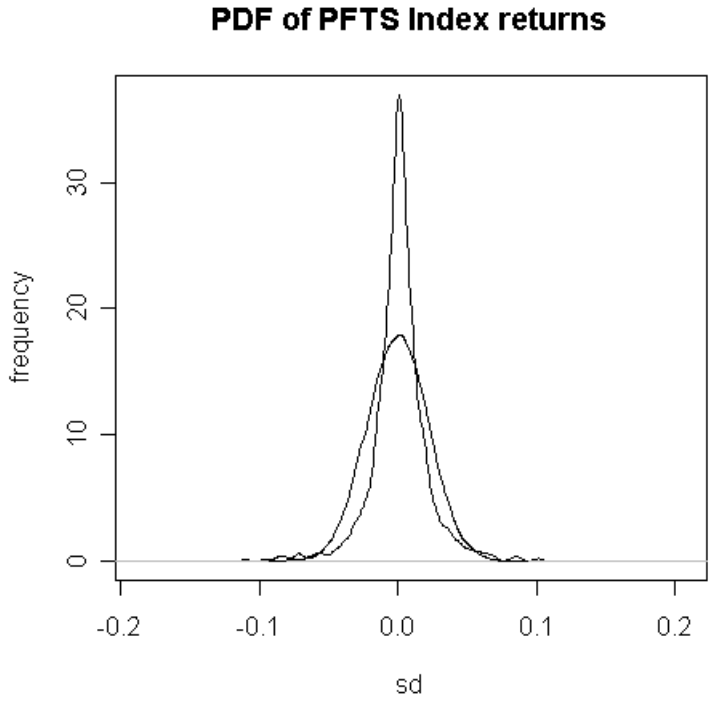

Figure 3.3. The estimated probability density function of PFTS daily returns accompanied by a normal distribution with the same mean and variance

Figure 3.4 demonstrates autocorrelation function (ACF) and partial autocorrelation function (PACF) of PFTS index returns. Autocorrelation function of a stationary process measures correlation between the process and its lags. Partial autocorrelation function also measures correlation between the process and its lag, but all intermediate lags eliminated from the process. This two functions helps researcher to identify the order of ARMA(p,q) process.

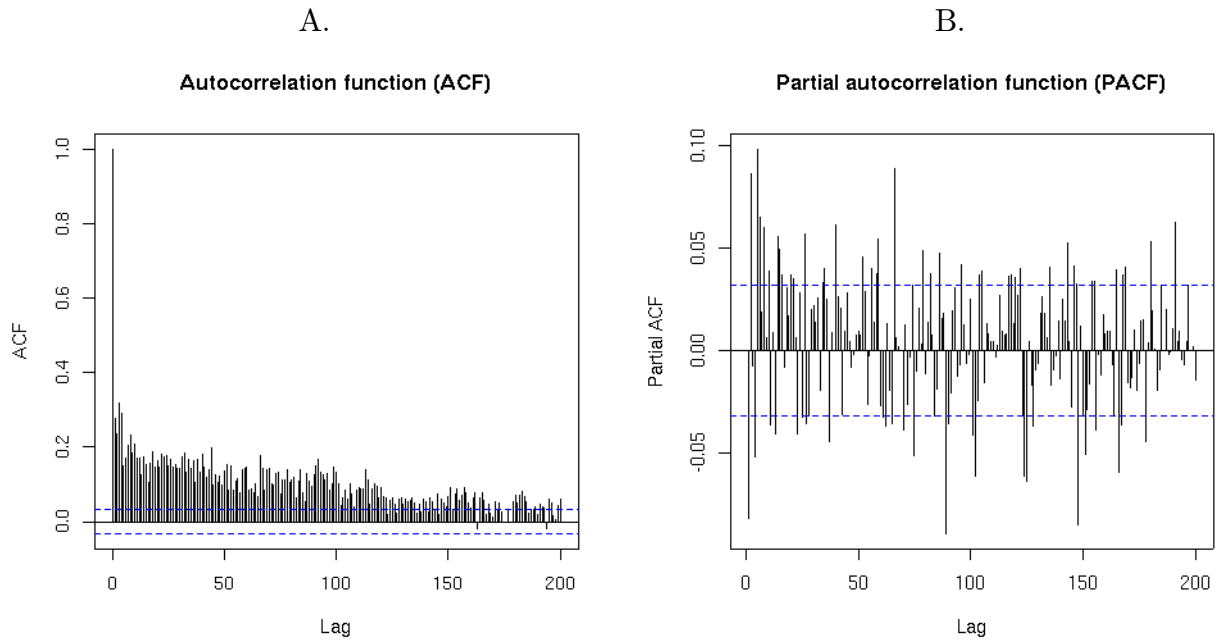

Figure 3.4. A. Autocorrelation function of PFTS absolute daily returns
B. Partial autocorrelation function of PFTS daily returns

Traditional ARMA processes have a short memory in the sense that the autocorrelation function decays exponentially. The autocorrelation function of a long memory process decays slowly. In fact, it decays so slowly that the autocorrelations are not summable:

$$\sum \rho(k) = \infty$$

Hence, the number of lags to include in the model is infinite. The figure 3.4A indicates that PFTS index daily returns is a long memory process. PFTS index daily returns ACF does not decay to zero even after 200 lags. Moreover, as figure 3.4B indicates the partial autocorrelation function of PFTS index returns also

does not decay to zero after 200 lags. Constantly, we can infer that PFTS returns is a long memory process.

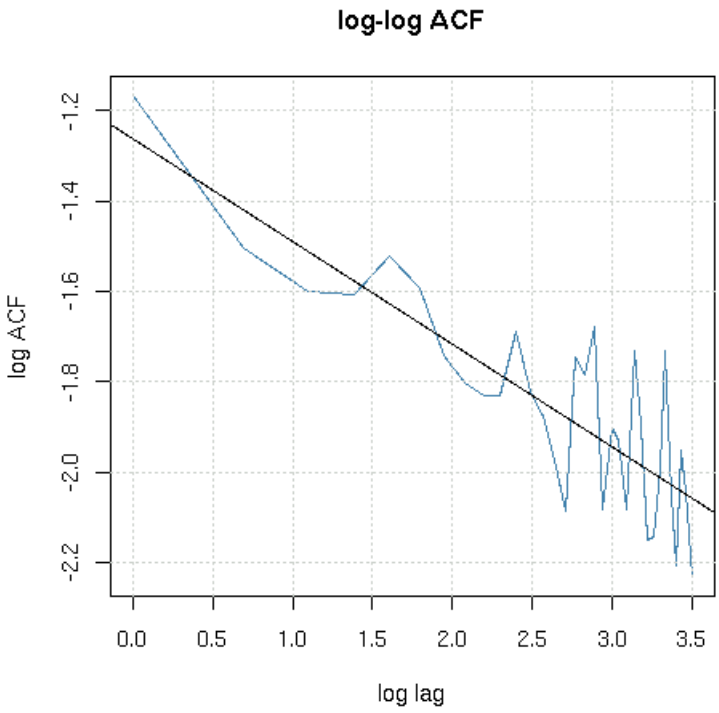

Figure 3.5. PFTS absolute daily returns the Hurst exponent estimation

Figure 3.5 shows estimated Hurst exponent(H) of PFTS daily returns. the Hurst exponent measures the degree of a long memory of the process. If the Hurst exponent is equal to $\frac{1}{2}$ then PFTS daily return follows regular random walk and does not have long memory, if $\frac{1}{2} < H < 1$ then PFTS daily return is a long memory process. The estimated Hurst exponent is equal to 0.8862, therefore PFTS daily return is a long memory process. the Hurst exponent of NYSE returns is also estimated for comparison and is equal to 0.7199. Hence, so far it was proved that PFTS index returns exhibits fractal properties, such as: long memory and fat-tailed distribution.

One property left, which attributed to the multifractal processes rather than to unifractal ones - change of the degree of long memory across time. This property is known as time rescale inconsistency. Unifractal models are incapable of

capturing this switching to different long memory degree. In order to check whether this property holds for Ukrainian PFTS returns, the data on PFTS returns was split into blocks by 2 years. the Hurst exponent was calculated for each block The results are presented in the table 3.3.

Period		PFTS returns Hurst exponent	NYSE returns Hurst exponent
1997-10-10	1998-12-31	0.6625	0.6728
1999-01-01	2000-12-31	0.8463	0.6025
2001-01-01	2002-12-31	0.5497	0.6967
2003-01-01	2004-12-31	0.7139	0.6561
2005-01-01	2006-12-31	0.7241	0.6246
2007-01-01	2008-02-11	0.8624	0.6711

Table 3.3. Hurst exponents blocked by 2 years

The table 3.3 indicates that the Hurst exponent of PFTS returns indeed varies of time. Also, the variation of long memory in PFTS stock exchange is higher than in case of developed market such as NYSE. Consequently, multifractal model is a better than unifractal choice for modelling asset returns on Ukrainian stock market. Moreover the increase of forecasting power of multifractal model compared to unifractal model must more noticeable in PFTS exchange then in NYSE as variation of long memory is considerable wider in Ukrainian market.

Chapter 4
Econometric model

The main assumptions of the model is that volatility can be described by equation (4.1), which is consistent with assumptions made by GARCH models. x_t is simple asset returns at period t, $x_t \equiv \frac{P_t - P_{t-1}}{P_{t-1}}$, where P_t is equal to asset price P_t at period t, error terms ε_t are i.i.d standard normal distribution $N(0,1)$

$$x_t = \sigma(M_{1t}M_{2t}...M_{kt})^{\frac{1}{2}}\varepsilon_t \tag{4.1}$$

The volatility components M_{kt} are assumed for simplicity to be statistically independent. The volatility components M_{kt} are modelled as states of Markov process. Each volatility component models the fractional random walk with different Hurst exponent H, therefore different degree of long memory.

The transition probabilities among states of Markov process are specified in [Calvet and Fisher 2004] as:

$$\gamma_k = 1 - (1 - \gamma_1)^{(b^{k-1})} \tag{4.2}$$

where $\gamma_1 \in (0,1)$ and $b \in (1, +\infty)$. k determines the number of volatility components, thus the number of volatility frequencies in the model. Therefore, k is considered to be a part of model selection problem. In this study k is equal to 15. This is compromise between desired forecasting power and computational feasibility.

The model is very flexible in specifying volatility components M_{kt}. Only two restrictions are applied to M_{kt}: $M_{kt} > 0$, $E(M_{kt}) = 1$. Hence, M_{kt} can be specified both parametrically and non-parametrically. In [Calvet and Fisher 2002] M_{kt} are specified as log normal distributed with mean arbitrary mean m_0 and unconditional variance σ equal to variance of asset returns in (4.1).

There are at least two methods available to estimate such models, either using maximum likelihood or generalised method of moments. Maximum likelihood requires the volatility components to follow discrete distribution and keep k small, as number of calculations increase exponentially 2^k. Generalised method of moments does not impose restrictions on parameters, but gives only linear forecasts.

The maximum likelihood method is too computationally intensive to simulate the whole year based on this method, hence GMM method is favoured in this study. According to [Lux 2006] the loss of efficiency caused by GMM can filled out by increase in number of volatility components. Estimation procedure is carried out as described in [Lux 2006] with out any deviations.

$k + 2$ parameters: $\gamma_1, \gamma_2, ..., \gamma_k, b$ plus M distribution characterising parameters must be estimated in MSM model. Hence we need at least $k + 2$ moment conditions to clearly identify parameters. Moments analytically solved and derived in [Lux 2006].

For more details about estimation procedure reader should consult to the original paper [Lux 2006].

Chapter 5
Model evaluation

5.1 Relation mean absolute deviation and mean square error

In order to evaluate the MSM forecasting accuracy and compare it to GARCH and FIGARCH models, the effective comparative criterion must be designed. Unfortunately, no single criterion is developed to fully characterise the out of sample predictive power of the estimator. The study [Hyndman and Koehler 2006] compares many widely used measures of predictive power, such as MAE, MSE, RMSE, MAPE, MdAPE, sMAPE, sMdAPE, MdRAE, GMRAE, MASE. This study presents persuasive arguments in support of relative measures of forecasting accuracy. Relative measures are not effected by units of measurement, easy to interpret and always exit. Relative measures compare the model to the benchmark model. Usually as benchmark is a naive forecast method, such as $x_t = x_{t-1}$ (next future value is equal to previous past value).

- relative MAE (mean absolute error)

$$\text{relative MAE}_T = \frac{\sum_{t=1}^{T} |\text{actual}_t - \text{predicted}_t|}{\sum_{t=1}^{T} |\text{actual}_t - \text{benchmark}_t|}$$

- relative MSE (mean square error)

$$\text{re lative MSE}_T = \frac{\sum_{t=1}^{T} (\text{actual}_t - \text{predicted}_t)^2}{\sum_{t=1}^{T} (\text{actual}_t - \text{benchmark}_t)^2}$$

These criteria relative MAE, MSE depend on time horizon T, hence must be computed for different value of T. However, the dependency of relative MAE, MSE on time horizon T may lead to impossibility of model comparison, for example for 5 day horizon GARCH(1,1) model can be better, but for 6 day MSM becomes better and for 7 day horizon vice versa. Fortunately, the estimates are consistent.

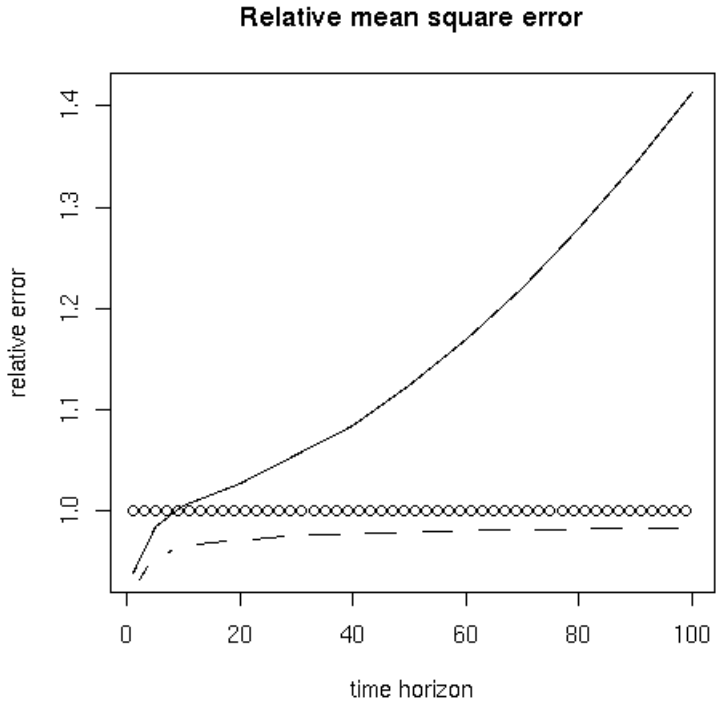

Figure 5.1. Out of sample relative mean square error.
Solid line - GARCH(1,1), dash line - MSM.

The figure 5.1 illustrate the forecasting power of GARCH(,1) and MSM models based on MSE. The results obtained are the average of 450 simulations on the PFTS returns of 2007 and first 2 months. The GARCH(1,1) model outperforms the naive forecast only at time horizon on less then 7 days. The accuracy of GARCH(1,1) forecast decreases quadratically. On the other hand, the accuracy of MSM model is systematically better than benchmark naive model. The MSM model constantly shows the relation MSE of MSM model holds in the range from 0.921 to 0.983. These results are consistent with results of [Lux 2006], with one

exception that in that study GARCH(1,1) is better for time horizon less then 3 days. In PFTS case, the MSM model is better at every time horizon.

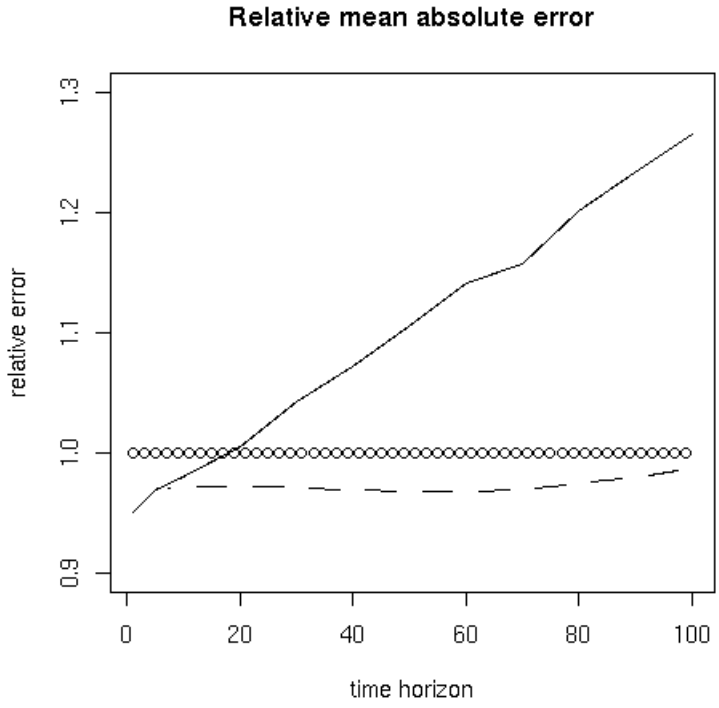

Figure 5.2. Out of sample relative mean absolute error. Solid line - GARCH, dash line - MSM.

Figure 5.2 demonstrates the out-of-sample relative mean absolute error. The mean absolute shows almost the same pattern as relative MSE. GARCH(1,1) model outperforms a naive benchmark only for time horizon less than 17 days. The forecasting accuracy of GARCH(1,1) falls linearly with increase of time horizon. MSM forecasting power steadily holds in range 0.956 to 0.987.

5.2 Portfolio optimization based on the forecast

The second criterion for evaluation and comparison of forecasting methods that is proposed in this study is amount of money earned relying on specific forecast method. This criterion has several advantages: clear economic interpretation, pos-

sibility of forecasting model comparison to simple investment strategies, like buy all stocks in equal proportions or proportionally to market capitalisation, which are very common among investors. Drawbacks of this criterion: difficult to implement as optimal stock trading strategy must be found relying on a price and volatility forecast and this criterion also depends on time horizon.

In this study we concentrate on simulation of 2007 year in PFTS stock market. The simulation is conducted in the following way:

1. For every day of 246 business days of 2007 repeat the following steps.
2. Forecast for every stock bid and ask prices using one of forecasting methods using all quoting history of this stock available up to considered day.
3. Solve a portfolio rebalance problem using forecasted expected returns and risk.
4. Start a new business day.
5. Calculate an earned return of the portfolio.
6. Go to the step 2.

The simulation is designed to be as realistic as possible thus a lot of factors must be taken into account:

— Transaction cost. A commission fee of 0.5% per transaction.

— Bid and ask spread

— 18 stocks plus risk-less asset, which is cash. Since Ukrainian bonds can't be considered as risk-less assets the cash plays the role of risk-less asset. In 2007 the inflation was -16.6% thus daily cash return is taken as -0.0456%

— Initial portfolio is risk-less portfolio of 1 million dollars

— Short selling of stocks is not allowed, according to the rules of PTFS stock exchange

5.2.1 Portfolio optimization setup

The class portfolio optimization problem by Markowitz is set in the following way

$$\begin{aligned} \max_{w_1,\ldots,w_n} U(E(r_p),\sigma_p), \qquad & \text{subject to} \\ \sigma_p &= \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \sigma_{ij} \\ E(r_p) &= \sum_{i=1}^{n} w_i E(r_i) \\ 0 \leqslant\ w_i &\leqslant 1, i = 1 \ldots n \\ \sum_{i=1}^{n} w_i &= 1 \end{aligned}$$

where

- w_i - weight of asset i in the portfolio
- $U(E(r_p),\sigma_p)$ - utility function of the particular investor
- $E(r_p)$ - expected return of the portfolio
- $E(r_i)$ - expected return of asset i
- σ_p - variance of the portfolio
- σ_{ij} - covariance between asset i and asset j
- n - number of assets

However, Markowitz portfolio optimization is based on the assumption of normally distributed returns of assets, therefore $E(r_i)$ and σ_i are fixed parameters. As it was shown in section 3.3, the normality assumption is violated on the PFTS stock exchange. The returns on Ukrainian stock returns follow fat-tailed distribution, which might have infinite variance, therefore another measure of risk must be introduced. Mean absolute deviation is another popular measure risk. According to [Feinstein and Thapa 1993], it makes optimization problem linear, robust to the outliers, does not require normal distribution, and the Mean-MAD portfolio optimization results are roughly equivalent to the results of Mean-Variance portfolio optimization. Hence, MAD is chosen as the risk measure.

Another problem with classical portfolio optimization approach is that investors utility function is undefined and investor specific. In this study the Sharpe ratio $\frac{E(r_p)}{\mathrm{MAD}_p}$ is taken as utility function. The Sharpe ratio [Sharpe 1966] is a common measure of portfolio performance. Of course is not ideal measure of portfolio performance, but since the goal of this paper is not find optimal investor's utility function, the most widespread function is used instead.

$$\begin{aligned}
\max_{w_1,\ldots,w_n} \frac{E(r_p)}{\mathrm{MAD}_p}, \qquad & \text{subject to} \\
\mathrm{MAD}_p &= \sum_{i=1}^{n} w_i \mathrm{MAD}_i \\
E(r_p) &= \sum_{i=1}^{n} w_i E(r_i) \\
0 \leqslant w_i &\leqslant 1, i = 1 \ldots n \\
\sum_{i=1}^{n} w_i &= 1
\end{aligned}$$

A few issues as still left with this specification: (i) transaction costs and bid-ask spread must be taken into account, as daily rebalances significantly reduces earned return, and (ii) with violation of normality assumption of asset returns, $E(r_i)$ and MAD_i are no longer constants over time. The last assumption is especially difficult to relax, since dependence of $E(r_i)$ and MAD_i on time can't be expressed by differentiable function. Consequently, such optimization problem can't be solved by conventional gradient optimization methods.

The final optimization problem specification

$$\begin{aligned}
\max_{w_1,\ldots,w_n,t_1,\ldots,t_n} \frac{E(r_p)}{\mathrm{MAD}_p}, \qquad & \text{subject to} \\
\mathrm{MAD}_p &= \sum_{i=1}^{n} w_i \mathrm{MAD}_i(t_i) \\
E(r_p) &= \sum_{i=1}^{n} w_i E(r_i(t_i)) \\
\sum_{i=1}^{n} t_i E(r_i(t_i)) - \sum_{i=1}^{n} t_i^0 E(r_i^0(t_i)) &\geqslant \mathrm{TC} + P_p^0 - P_p \\
0 \leqslant w_i &\leqslant 1, i = 1 \ldots n \\
\sum_{i=1}^{n} w_i &= 1
\end{aligned}$$

where

- TC is transaction costs
- $P_p^0 - P_p$ is difference of bid prices between desired portfolio and initial portfolio (loss occurred due to bid-ask spread)
- $n = 19$. PFTS index consists of 18 stocks plus risk-less asset.
- t_i - investment horizon for the asset i, measured in days.
- $E(r_i(t_i))$ - expected return of asset i over next t_i days
- $\text{MAD}_i(t_i)$ - mean absolute deviation of asset i over next t_i days
- $t_i E(r_i(t_i))$ - total expected return of asset i

5.2.2 Choice of the optimization algorithm

In order solve portfolio optimization problem set up in previous subsection and find optimal stock trading strategy, the complex optimization problem in discrete time must be solved. There are four broad families of optimization algorithms:

— Gradient optimization methods are hardly applicable for the search of optimal investment strategy, because they require continuous differentiable function for optimization and portfolio optimization problem is defined on discrete. No particular function form can be assumed for neither expected return nor risk Therefore the optimization problem is partially combinatorial and can't be represented as continuous function optimization.

— Exhaustive search, it requires for N stock at least $N^{C_N^1+C_N^2+...+C_N^N} = N^N$ operations, roughly speaking, then for 18 stocks we have $\approx$ 3.9e23 operations that is computationally unfeasible. Current top supercomputer makes 480e12 operations per second, therefore such algorithms would require 2.5 years to finish.

— Genetic optimization methods (simulated annealing, simulated tempering, coevolution and etc.), this methods suite the best, they require neither continuous differentiable function nor exhaustive search.

— Dynamic programming are applicable, but with several simplifying assumptions must be introduced, one of which is the absence of the long memory. Such assumption is not definitely undesirable, as this is a subject of our study.

Hence the portfolio optimization problem will be solved using genetic optimization algorithm.

5.2.3 Genetic portfolio optimization algorithm

Genetic algorithm solves two problems: a search of optimal combination of assets in the portfolio and search of most profitable investment horizon for each stock. In order to differentiate a notion used in this study, the asset is treated differently from a stock. The asset is stock with predetermined investment horizon, therefore a stock held for 3 days is an asset, but the same stock held for 5 days is a completely different asset. As genetic algorithm looks for optimal combination of assets, it has to choose among $246^{18} \approx 1e43$ possible assets, as there were 246 business days in 2007.

The genetic algorithm for portfolio optimization was developed on the base of regular coevolution algorithm. The terminology used in this study coincides with the one in the book [Chambers 2001]. If reader is not familiar with genetic algorithms and terminology of evolutionary programming, he can consult the section 1.4 of the book.

Coding scheme for portfolio optimization used in this study:

– Population size is taken by 1000 chromosomes on each island

- Chromosome is portfolio of 19 genes, assets (18 stocks and cash)
- Gene is one of an assets (18 stocks and cash with attached investment horizon)
- Fitness function, the most fitted portfolio (chromosome) is portfolio with highest Sharpe ratio ($\frac{\text{expected return}}{\text{risk}}$) subject to transaction cost constraint
- Random chromosome, is a random portfolio with random weights of assets with random investment horizon from 1 to number of days left in simulation
- Crossover function. The single point cross over function, in order to eliminate the possibility of getting unfeasible portfolio after crossover, the weighted average of two portfolios was used.
- Crossover rate is equal 40%
- Selection function. The generic selection function was used - tournament with tournament size equals to 4
- Number of islands is equal to 4
- Migration rate among islands is 10 chromosomes per 4 generations
- Migration topology is complete, hence migration happens between all islands
- Mutation rate is 15%

The performance of genetic optimization algorithm crucially depends on values of control parameters: population size, number of islands, crossover rate, migration rate, migration topology and mutation rate. In order to calibrate the algorithm or more precisely find appropriate values for control parameters as the forecasting values for stock returns were taken a true future values. Values of control para-

meters were found values by means of trial error that give steady optimum and fast convergence.

The genetic algorithm was implemented in C++ programming language with the help of GALib developed at MIT, freely available at http://lancet.mit.edu/ga. The Source code for optimization algorithm is provided in appendix B.

5.2.4 Results

The simulation results for 2007 are presentable in table 5.1. The second column presents results when only commission fee 0.5% per transaction is taken into account. The third column represents results when both commission fee and bid-ask spread is taken into account. The genetic algorithm is stochastic optimization method that this stopped when some tolerance level achieved, that is why the final results contain small discrepancy.

The passive investment strategy that follows PFTS Index has brought in return 134%. Perfect forecast was use to calibrate the optimization algorithm. The results portfolio optimization based on perfect forecast are impressive. They are hardly believable, but it is true. 2007 was very successful year in the Ukrainian stock market, some stocks prices increased in 11 times, others - 6 times thus 1760% return was indeed reachable in 2007. The portfolio optimization based on perfect forecast is done rather to evaluate the quality of optimization algorithm.

The portfolio optimization results based on one of the forecasting methods are much more modest then results based on perfect forecast. They a close to the passive investment strategy. GARCH(1,1) model was capable to outperform the market only in case no bid-ask spread. Addition of bid-ask spread lowered the GARCH(1,1) return to 129%. On the contrary, the MSM model outperformed market considerably by 53% in case of transaction costs, and by 27.35% in case of transaction costs and bid-ask prices.

Forecasting method	Annual return Transaction costs, quote prices	Annual return Transaction costs, bid-ask prices
Passive investment strategy	135.8%	134%
Perfect forecast	$1913\% - 1982\%$	$1734\% - 1760\%$
GARCH	$147.3\% - 150.1\%$	$127.3\% - 130.5\%$
MSM	$187.2\% - 190.9\%$	$159.8\% - 162.9\%$

Table 5.1. Portfolio optimization results

Also it can be note that even small transaction costs reduce earned return considerably for active investors. Bid-ask spread in 2007 was around 2%, this 2% led to fall in annual return by at least 20%.

Chapter 6
Conclusions

This paper investigates closely the stock returns of transition countries, such as Ukraine. The stock markets of transition countries is much more volatile than stock markets of developed countries. This makes conventional econometric models, like GARCH to perform poor forecast in transition countries. This paper showed that PFTS stock returns exhibits such fractal properties:

- long memory, the infinite lag length;
- lags decay hyperbolically, slower than exponentially;
- returns to assets follow not normal fat-tailed distribution;
- different degree of long memory, which known as time rescaling inconsistency.

Therefore, the multi-fractal model is proposed capture all found properties in transition countries. All of this properties can be attributed to the developed countries as well. However, the last property suites specifically to transition countries, as it turned out the change of long memory degree happens more frequently in transition stock markets than in developed countries. Also the variety of long-memory degrees is much wider in transition countries such as Ukraine. Therefore, multifractal model is a good choice for modelling the asset returns in transition countries.

The first application of MSM models in the transition countries stock markets in this study revealed that the increase of forecasting accuracy by MSM model is tremendous compared to classical GARCH(1,1) model. The performance of GARCH(1,1) is really poor in Ukrainian PFTS stock market, as it was shown in section 3.3. No wonder that no Ukrainian investment companies use it to forecast stock returns. The MSM model is a huge step ahead. The MSM forecast showed that it consistently better that naive benchmark model. Although, the MSM model was only 8% better than naive benchmark model it was enough to outperform the market return.

As portfolio optimization bases on the GARCH and MSM forecasts gave a economic justification of forecast accuracy. the MSM model allowed to outperform the PFTS market on average by 27.35% in 2007, while GARCH model annual return showed the return of 6% lower than the market. It is incredible finding. Further studies are necessary to check whether other markets than Ukraine will show similar results.

Bibliography

[Baillie *et al.* 1996] R. T. Baillie, T. Bollerslev, and H. O. Mikkelsen. Fractionally intergrated generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 74, 1996.

[Bollerslev 1986] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31, 1986.

[Calvet and Fisher 2001] Laurent Calvet and Adlai Fisher. Forecasting multifractal volatility. *Journal of Econometrics*, 44, 2001.

[Calvet and Fisher 2002] Laurent Calvet and Adlai Fisher. Multifractalily in asset returns: Theory and evidence. *The Review of Economics and Statistics*, 31, 2002.

[Calvet and Fisher 2004] Laurent Calvet and Adlai Fisher. *Regime-Switching and the Estimation of Multifractal Processes*. Harvard University. Cambridge, Massachusetts, 2004.

[Chambers 2001] Lance D. Chambers. *The practical handbook of genetic algortihms, Applications*. Chapman & Hall/CRC, 2001.

[Engle 1982] R. F. Engle. Autoregressive conditional heteroskedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50, 1982.

[Falconer 1985] K. Falconer. *Geometry of Fractal Sets*. Cambridge University Press, 1985.

[Feinstein and Thapa 1993] Charles D. Feinstein and Mukund N. Thapa. A reformulation of a mean-absolute deviation portfolio optimization model. *Management Science*, 39:1552–1553, 1993.

[Hyndman and Koehler 2006] Rob J. Hyndman and Anne B. Koehler. Another look of measures of forecast accuracy. *International Journal of Forecasting*, 22:679–688, 2006.

[Kaye 1989] Brian Kaye. *A Random Walk Through Fractal Dimensions*. VCH Publishers, New York, 1989.

[Lux and Kaizoji 2006] Thomas Lux and Taisei Kaizoji. *Forecasting Volatility and Volume in the Tokyo Stock Market: Long Memory, Fractality and Regime Switching*. PhD thesis,

Department of Economics. University of Kiel, Olshausenstr. 40, 24118 Kiel, Germany, 2006.

[Lux 2006] Thomas Lux. *The Markov-Switching Multifractal Model of Asset Returns: GMM Estimation and Linear Forecasting of Volatility*. Department of Economics. University of Kiel, Olshausenstr. 40, 24118 Kiel, Germany, 2006.

[Mandelbrot *et al.* 1996] Benoit Mandelbrot, Adlai Fisher, and Laurent Calvet. *A Multifractal Model of Asset Returns*. Department of Mathematics, Yale University and IBM T. J. Watson Research Center, October 1996.

[Mandelbrot 1977] Benoit Mandelbrot. *The Fractal Geometry of Nature*. W.H. Freeman and Company, New York, 1977.

[Perevozchikov 2007] Vladislav Perevozchikov. *Which Companies Go Public in Ukraine*. Economics education and research consortium, 2007.

[Peters 1991] Edgar E. Peters. *Fractal Market Analysis. Applying Chaos Theory to Investment and Economics*. John Wiley & Sons, Inc, 1991.

[Peters 1994] Edgar E. Peters. *Chaos and Order in the Capital Markets*. John Wiley & Sons, Inc, 1994.

[R Development Team 2006] R Development Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[Sharpe 1966] William F. Sharpe. Mutual fund performance measurement. *Journal of Bussiness*, pages 118–138, 1966.

[Taylor 1986] S. Taylor. *Modelling Financial Time Series*. John Wiley & Sons, Inc, 1986.

Appendix A

Listing of plotting graphs for section 3.3 in *R* statistical package [R Development Team 2006]

```
   R version 2.6.0 (2007-10-03)
   Copyright (C) 2007 The R Foundation for Statistical Computing
> library(foreign)
> library(fBasics)
> library(zoo)
> data = read.dbf("pfts_index_cleaned.dbf")
> pfts = zoo(data[[2]], order.by=data[[1]])
> plot(pfts, main="PFTS Index", xlab="time", ylab="index")
> returns = diff(log(pfts))
> plot(returns*100, main="PFTS daily returns", xlab="time",
  ylab="%")
> acf(abs(returns), lag.max=200, na.action = na.pass,
  main="Autocorrelation function (ACF)")
> lmacfPlot(abs(returns))
   Long Memory Autocorrelation Function:
     Maximum Lag         33
     Cut-Off ConfLevel   0.03936496
     Plot-Intercept      -1.260971
     Plot-Slope          -0.2275325
```

```
     Hurst Exponent     0.8862338
```

```
> library(fEcofin)
> data(nyse)
> dates = as.character(nyse[,1])
> n = zoo(nyse[,2], order.by=as.Date(dates))
> plot(n)
> returns = diff(log(n))
> plot(100*returns, ylim=c(-20,20), xlab="time", ylab="%")
>
```

Appendix B

Genetic optimization algorithm developed in C++ language. In order to build this program all four files (galib.h, galib.cpp, blas.h, logstreambuf.h) and galib, boost and R libraries are required. Library Galib is available at http://lancet.mit.edu/ga. Boost is available at http://boost.org. *R* is available at http://www.r-project.org

B.1 File galib.h

```
#include <ga/GAGenome.h>
#include <ga/GA1DArrayGenome.h>


struct gene_t {
    double w;           // weight of stock
    double q;           // quantity of stock
    int time;           // stock holding time
    gene_t() {}
    gene_t(int x) {}
    gene_t(const gene_t &t) : w(t.w), q(t.q), time(t.time)
    {}

};

bool operator==(const gene_t &g1, const gene_t &g2) {
        if (g1.w == g2.w && g1.time == g2.time && g1.q == g2.q)
            return true;
        else
            return false;
}

bool operator!=(const gene_t &g1, const gene_t &g2) {
        return !(g1 == g2);
}
```

```
   void my_random_chromosome(GAGenome &);
   int my_mutate_chromosome(GAGenome &, float);
   int my_crossover(const GAGenome &, const GAGenome &, GAGenome *,
GAGenome *);
   float my_fitness(GAGenome &);


   template <class T>
   class MyGenome : public GA1DArrayGenome<T> {
       public:
       GADefineIdentity("MyGenome", 201);

       MyGenome(int s, float (*fit) (GAGenome &)):
                   GA1DArrayGenome<T>(s, fit)
       {
           initializer(Init);
           mutator(Mutate);
           crossover(Cross);
       }

       static void Init(GAGenome &g) {
           return my_random_chromosome(g);
       }

       static int Mutate(GAGenome &g, float f) {
           return my_mutate_chromosome(g, f);
       }
       static int Cross(const GAGenome &p1, const GAGenome &p2,
GAGenome *c1, GAGenome *c2) {
           return  my_crossover(p1, p2, c1, c2);
       }

       static float Evalute(GAGenome &g) {
           return  my_fitness(g);
       }

       const T& operator [] (int x) const {
           return this->a[x];
       }

        T& operator [] (int x) {
           return this->a[x];
       }
   };


   /// Forward declarations
   typedef MyGenome<gene_t>        genome_t;

   void print_chromosome(genome_t &, const char*, const char*, int);
   void fill_in_quantities(genome_t &, double *, double *);
```

B.2 File galib.cpp

```
   #include "blas.h"
   #include "logstreambuf.h"
   #include "galib.h"
   #include <boost/random.hpp>
   #include <ga/ga.h>
   #include <assert.h>
   #include <time.h>
   #include <iomanip>
   #include <cstdlib>
   #include <cstring>
   #include <streambuf>
   #include <ctime>
   #include <algorithm>
   #include <numeric>
   #include <fstream>
   #include <iostream>
   #include <iterator>
   #include <deque>

   using namespace std;
   using namespace boost;
   using namespace boost::numeric::ublas;

   extern "C" {
       #include<R.h>
       #include<Rinternals.h>
   }

   #define MAX_NUM_THREADS 4 // number of threads
   #define L 19 // no. of genes in a chromosome (18 stocks + cash)
   #define POPULATION_SIZE 700  // no. of chromosomes in population
   #define MAX_SIM    100 //no of simulation done for each asset
   #define MUTATION_RATE   0.3
   #define MUTATION_STD    0.3 // Standard deviation of normally
distributed random mutation
   #define CONVERGENCE_RATE  0.99
   #define CONVERGENCE_GENERATIONS  30 // Number of stable
generations to stop optimization
   #define MIN_GENERATIONS         20
   #define MAX_GENERATIONS         1000
   #define CROSSOVER_RATE  0.9
   #define GENETIC_SIMULATION      10

   /// Seed the random numbers generators
   static lagged_fibonacci2281 rng_real(time(NULL));
```

```
   static mt11213b rng(time(NULL));
   uniform_01<mt11213b> unif01_random(rng);

   double commission_fee = 0.005; // 0.5% per transaction

   double initial_portfolio[L] = {1000000, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0,
                                  0, 0, 0, 0};

   double initial_portfolio_weights[L] = {1, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0,
                                  0, 0, 0, 0};

   double initial_portfolio_price = 1000000;
   double initial_portfolio_total_expected_return = -10;

   int holding_time_left = 245;

   int holding_time[L] = {245, 1, 1, 1, 1,
                       1, 1, 1, 1, 1,
                       1, 1, 1, 1, 1,
                       1, 1, 1, 1};

   double current_bidprices[L] = {1, 2, 3, 4, 5,
                             6, 7, 8, 9, 10,
                             11, 12, 13, 14, 15,
                             16, 17, 18, 19
                               };

   double current_askprices[L] = {1, 3, 4, 5, 6,
                             7, 8, 9, 10, 11,
                             12, 13, 14, 15, 16,
                             17, 18, 19, 20
                               };

   matrix3d *simulated_future;
   matrix2d *simulated_returns;
   matrix2d *simulated_risks;

   // fitness evaluation function
   float my_fitness(GAGenome &genome)
   {
           genome_t &chromosome = (genome_t &) genome;

           print_chromosome(chromosome, __FILE__, __func__,
__LINE__);

           double portfolio_price = 0;
           double transaction_costs = 0;

           fill_in_quantities(chromosome, &portfolio_price,
&transaction_costs);
```

```
            // Simulate portfolio to get mean return and risk
            vector1d asset_returns(L);
            vector1d asset_risks(L);

            double portfolio_return = 0;
            double portfolio_risk = 0;

            print_chromosome(chromosome, __FILE__, __func__,
__LINE__);


            for(int i=0; i < L; i++) {
                if(chromosome[i].time >= 1) {
                    asset_returns[i] = (*simulated_returns)(i,
chromosome[i].time - 1);
                    portfolio_return += chromosome[i].w *
asset_returns[i];

                    asset_risks[i] = (*simulated_risks)(i,
chromosome[i].time - 1);
                    portfolio_risk += chromosome[i].w *
asset_risks[i];
                }
                else {
                    asset_returns[i] = 0;
                    asset_risks[i] = 0;
                }
            }


            double total_expected_return = 0;
            for(int i=0; i < L; i++)
                total_expected_return += chromosome[i].time *
asset_returns[i] * chromosome[i].q;


            float fit;

            portfolio_risk = max(1e-25, portfolio_risk);
            if(total_expected_return -
initial_portfolio_total_expected_return < transaction_costs)
                fit = 10*(total_expected_return -
initial_portfolio_total_expected_return -  transaction_costs);
                //fit = -1e20;
            else
                fit = portfolio_return / portfolio_risk;

            assert(!isnan(fit));
            assert(!isinf(fit));
            return fit;

    }

    /*********
```

```
   * Fill in quantities from weights
   * returns transaction costs
   **********/
   void fill_in_quantities(genome_t &chromosome, double *p_price =
NULL, double *t_cost = NULL) {


       double q[L], q1[L], qdiff[L], buy[L], sell[L];
       double precision = 9e99;
       double transaction_costs, commission;
       double portfolio_price = initial_portfolio_price;
       int counter = 0;

       while(precision > 1e-8) {
           ++counter;

           transaction_costs = 0;
           commission = 0;

           for(int i=0; i < L; i++) {
               q[i] = chromosome[i].w * portfolio_price /
current_bidprices[i];
               qdiff[i] = q[i] - initial_portfolio[i];
               buy[i] = max(0.0, qdiff[i]);
               sell[i] = -min(0.0, qdiff[i]);

               transaction_costs += buy[i] * current_askprices[i] -
sell[i] * current_bidprices[i];
               commission += (buy[i] * current_askprices[i] +
sell[i] * current_bidprices[i]) * commission_fee;
           }

           portfolio_price = (portfolio_price +
initial_portfolio_price - transaction_costs - commission) / 2;

           precision = 0;
           for(int i=0; i < L; i++) {
               q1[i] = chromosome[i].w * portfolio_price /
current_bidprices[i];
               precision += (q[i] - q1[i])*(q[i] - q1[i]);
           }
           precision = sqrt(precision) / L;

           assert(counter < 100);
       }


       for(int i=0; i < L; i++)
           chromosome[i].q = q1[i];

       if(p_price)
           *p_price = portfolio_price;
       if(t_cost)
           *t_cost = transaction_costs + commission;
```

```
   }

   // random chromosome creation function
   void  my_random_chromosome(GAGenome &genome) {

       genome_t &chromosome = (genome_t &)genome;


       int assets[L] =
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};
       double weights_left = 1;
       double weight;
       uniform_int<> hold_dist(1, holding_time_left);

       random_shuffle(assets, assets+L);

       for(int *iter=assets; iter != assets + L - 1; iter++) {
           uniform_real<> weights_dist(0, weights_left);
           weight = weights_dist(rng);
           weights_left -= weight;
           chromosome[*iter].w = weight;

           // Generate holding periods
           chromosome[*iter].time = hold_dist(rng);
           chromosome[*iter].q = -1.0;
       }

       // The last asset is treated differently to be sure that
sum(assets_weights)=1
       chromosome[assets[L - 1]].w = weights_left;
       chromosome[assets[L - 1]].time = hold_dist(rng);
       chromosome[assets[L - 1]].q = -1.0;

       print_chromosome(chromosome, __FILE__, __func__, __LINE__);

   }

   // chromosome mutation function
   int my_mutate_chromosome(GAGenome &genome, float pmut) {

       genome_t &chromosome = (genome_t &)genome;

       print_chromosome(chromosome, __FILE__, __func__, __LINE__);

       double mutate_prob = unif01_random();

       if(mutate_prob > pmut) {
           return 0;
       }

       double change_genes = unif01_random();
       int indexes[L] =
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};
```

```
        random_shuffle(indexes, indexes+L);
        int index = indexes[0];

        /// Change either weight or holding time
        if(change_genes < 0.5) {

            double weight = -1;
            double weight_diff;

            normal_distribution<> ndist(chromosome[index].w,
MUTATION_STD);

            while( weight < 0 || weight > 1)
                weight = ndist(rng_real);

            weight_diff = weight - chromosome[index].w;
            chromosome[index].w = weight;

            double tmp;
            for(int i=1; i < L && weight_diff !=0; i++) {
                gene_t &cur_gene = chromosome[indexes[i]];
                if(weight_diff > 0)
                    tmp = min(cur_gene.w, weight_diff);
                else
                    tmp = max(weight_diff, cur_gene.w - 1);
                cur_gene.w -= tmp;
                weight_diff -= tmp;
            }
        }
        else {
            uniform_int<> hold_dist(1, holding_time_left);
            chromosome[index].time = hold_dist(rng);
        }

        print_chromosome(chromosome, __FILE__, __func__, __LINE__);
        return 1;
    }


    int my_crossover(const GAGenome &genome1, const GAGenome
&genome2, GAGenome *c1, GAGenome *c2) {

        genome_t &parent1 = (genome_t &)genome1;
        genome_t &parent2 = (genome_t &)genome2;
        genome_t child1(L, my_fitness);
        genome_t child2(L, my_fitness);


        double l=0;
        uniform_real<> lambda_dist(0.5, 1.0);
        l = lambda_dist(rng);

        for(int i=0; i < L; i++) {
            child1[i].w = l * parent1[i].w + (1-l) * parent2[i].w;
```

```
            child1[i].time = round((l * parent1[i].time + (1-l) *
parent2[i].time));
            child1[i].q = parent1[i].q;
            if(child1[i].time == 0) child1[i].time++;


            child2[i].w = l * parent2[i].w + (1-l) * parent1[i].w;
            child2[i].time = round((l * parent2[i].time + (1-l) *
parent1[i].time));
            child2[i].q = parent2[i].q;
            if(child2[i].time == 0) child2[i].time++;
        }

        int num=0;
        if(c1) {
            *c1 = child2;
            print_chromosome(child2, __FILE__, __func__, __LINE__);
            num++;
        }
        if(c2) {
            *c2 = child1;
            print_chromosome(child1, __FILE__, __func__, __LINE__);
            num++;
        }

        return num;
    }

    static int ga_restart = 0;

    GABoolean stop_criteria(GAGeneticAlgorithm &ga) {
        static int gen_counter = 0;
        GABoolean pop_term =
GAGeneticAlgorithm::TerminateUponPopConvergence(ga);
        if(pop_term && gen_counter > MIN_GENERATIONS)
            return pop_term;
        else if(gen_counter > MAX_GENERATIONS) {
            gen_counter = 0;
            ga_restart = 1;
            return GABoolean(1);
        }
        gen_counter++;
        return GABoolean(0);
    }

    extern "C" {
    void genetic_optim(double *initial_portfolioR, int
*holding_timeR, int *holding_time_leftR,
                       double *current_bidpricesR, double
*current_askpricesR,
                       double *simulated_futureR,
                       double *ret_portfolio_qtys, int
*ret_portfolio_times) {
```

```
       srand(time(NULL));
       genome_t    init_port(L, my_fitness);

       memcpy(initial_portfolio, initial_portfolioR, sizeof(double)
* L);
       memcpy(holding_time, holding_timeR, sizeof(int) * L);
       memcpy(current_askprices, current_askpricesR, sizeof(double)
* L);
       memcpy(current_bidprices, current_bidpricesR, sizeof(double)
* L);
       holding_time_left =  *holding_time_leftR;

       initial_portfolio_price = 0.0;
       for(int i=0; i < L; i++) {
           initial_portfolio_price += current_bidprices[i] *
initial_portfolio[i];
       }

       for(int i=0; i < L; i++) {
           initial_portfolio_weights[i] = current_bidprices[i] *
initial_portfolio[i] / initial_portfolio_price;
           init_port[i].w = initial_portfolio_weights[i];
           init_port[i].time = holding_time[i];
           init_port[i].q = initial_portfolio[i];
       }

       print_chromosome(init_port, __FILE__, __func__, __LINE__);

       cout << "N.AHEAD  " << holding_time_left
               << " Initial port price " << setw(7) <<
setprecision(5) << initial_portfolio_price << endl;

       matrix3d    sim_future(L);
       matrix2d    *mat[L];

       for (int i=0; i < L; ++i) {
           mat[i] = new matrix2d(holding_time_left, MAX_SIM);
           sim_future(i) = *mat[i];
       }

       cout << "Simulated future" << endl;
       for (int k=0; k < MAX_SIM; k++)
           for (int j=0; j < holding_time_left; j++)
               for (int i=0; i < L; i++)
                   sim_future(i)(j, k) = *(simulated_futureR +
k*L*holding_time_left + j*L + i);

       simulated_future = &sim_future;


       /*********
       * Calculate daily Expected returns and  Absolute deviations
(MAE) of assets
       **********/
```

```
        matrix2d sim_f(L, holding_time_left);
        matrix2d sim_returns(L, holding_time_left);
        matrix2d sim_risks(L, holding_time_left);

        for(int i=0; i < L; i++) {
            for(int j=0; j < holding_time_left; j++) {
                matrix_row<matrix2d> row(sim_future(i), j);
                sim_f(i,j) = sum(row) / MAX_SIM;
                sim_risks(i,j) = norm_1(row -
scalar_vector1d(MAX_SIM, sim_f(i,j)));

                if(j != 0) {
                    sim_returns(i,j) = (sim_returns(i,j-1) * j +
sim_f(i,j)) / (j+1);
                    sim_risks(i,j) = (sim_risks(i,j-1) * j +
sim_risks(i,j)) / (j+1);
                }
                else {
                    sim_returns(i,j) = sim_f(i,j);
                }
            }
        }

        initial_portfolio_total_expected_return = 0;
        for(int i=0; i < L; i++) {
            if(holding_time[i] >= 1 && holding_time[i] <=
holding_time_left)
                initial_portfolio_total_expected_return +=
                            initial_portfolio[i] * sim_returns(i,
holding_time[i]-1) * holding_time[i];
        }
        simulated_future = &sim_future;
        simulated_returns = &sim_returns;
        simulated_risks = &sim_risks;


        cout << "Start optimization" << endl;
        GARandomSeed(time(NULL));
        genome_t genome(L, my_fitness);

        GASteadyStateGA ga(genome);
        GASigmaTruncationScaling scal;

        ga.pCrossover(CROSSOVER_RATE);
        ga.pMutation(MUTATION_RATE);
        ga.pConvergence(CONVERGENCE_RATE);
        ga.nConvergence(CONVERGENCE_GENERATIONS);
        ga.terminator(stop_criteria);
        GATournamentSelector        select;
        ga.selector(select);
        ga.crossover(my_crossover);
        ga.scaling(scal);
```

```
        ga.populationSize(POPULATION_SIZE);
        ga.nReplacement( POPULATION_SIZE / 4);
        ga.nGenerations(500);

        genome_t best(L, my_fitness);
        float fit;
        int counter = 0;
        do {
            counter++;
            ga_restart = 0;
            cout << "Try " << counter << endl;
            ga.evolve(time(NULL));
            best = (genome_t &) ga.statistics().bestIndividual();
            fit = my_fitness(best);
            cout << "Fit " << fit << " Total gens " <<
ga.generation() << endl;
        } while(fit < -10 && counter < 100 && !ga_restart);



        cout << "TOTAL GENERATIONS " << ga.generation() << endl;
        cout << "POPULATION CONVERGENCE " <<
ga.statistics().convergence() << endl;
        cout << "BEST PORTFOLIO" << endl;
        print_chromosome(best , __FILE__, __func__, __LINE__);
        cout << "Best portfolio utility "<< fit << endl;


        float init_fit = my_fitness(init_port);
        if(fit < init_fit || counter == 100) {
            // Initial portfolio is the best portfolio
            best = init_port;
        }

        ga.flushScores();


        for(int i=0; i < L; i++) {
            ret_portfolio_qtys[i] = best[i].q;
            ret_portfolio_times[i] = best[i].time;
        }

        for(int i=0; i < L; ++i)
            delete mat[i];

        cout << "Genetic optim ended" << endl;
    }
    }

    void print_chromosome(genome_t &chromosome, const char *FILE,
const char *func, int LINE_NUM) {

        double sum=0.0;
        double price=0.0;
```

```
        logstreambuf buffer("dummy");
        ostream     cout(&buffer);

        cout << "Executed by "<< FILE << " line " << LINE_NUM << "
function " << func << endl;

        cout << "weights" << endl;
        for (int j=0; j < L; ++j) {
            cout << setw(7) << setprecision(2) << chromosome[j].w;
            sum += chromosome[j].w;
            assert(chromosome[j].w >= 0 && chromosome[j].w <= 1);
        }

        cout << endl << "quantities" << endl;
        for (int j=0; j < L; ++j) {
            cout << setw(5) << chromosome[j].q;
            price += chromosome[j].q * current_bidprices[j];
            assert(chromosome[j].q >= 0 || chromosome[j].q == -1);
        }

        cout << endl << "portfolio price " << setprecision(5) <<
price
                << " initial_price " << setprecision(5) <<
initial_portfolio_price << endl;
        cout << "hold time" << endl;
        for (int j=0; j < L; ++j) {
            cout << setw(5) << chromosome[j].time;
            assert(chromosome[j].time >= 0 && chromosome[j].time <=
holding_time_left);
        }
        cout << endl << "Sum of weights =" << sum << endl;

        cout << "Executed by "<< FILE << " line " << LINE_NUM << "
function " << func << endl;
        assert(abs(sum - 1.0) < 1.e-13);
    }
```

B.3 File blas.h

```
    #include <boost/numeric/ublas/vector.hpp>
    #include <boost/numeric/ublas/matrix.hpp>
    #include <boost/numeric/ublas/matrix_proxy.hpp>
    #include <boost/numeric/ublas/io.hpp>

    using namespace boost::numeric::ublas;

    typedef vector<int>     vector_int;
```

```
typedef vector<double>  vector1d;
typedef scalar_vector<double>   scalar_vector1d;
typedef matrix<double>  matrix2d;
typedef vector<matrix2d> matrix3d;
```

B.4 File logstreambuf.h

```
#ifndef LOGSTREAMBUF_H
#define LOGSTREAMBUF_H
#include<string>
#include<streambuf>
#include<iostream>
#include<syslog.h>

using namespace std;
class logstreambuf : public streambuf
{
    public:
         logstreambuf(const char *appname) : streambuf()
         {
            openlog(appname, LOG_PID | LOG_CONS, LOG_DAEMON);
            setp(buffer_, buffer_ + bufsize_);
         }
         virtual ~logstreambuf()
         {
            atexit(closelog);
         }

         virtual int sync()
         {
            string tmp(pbase(), pptr());
            syslog(LOG_INFO, tmp.c_str());
            pbump( - (pptr() - pbase()));
            return 0;
         }
    private:
        static  const   size_t  bufsize_ = 1024;
        char    buffer_[bufsize_];
};

#endif
```