Extreme Values of Gaussian Processes and A Heterogeneous Multi Agents Model

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Christoph Manuel Schmid

von Niedermuhlern/BE

Leiter der Arbeit: Prof. Dr. J. Hüsler
Institut für mathematische Statistik
und Versicherungslehre
Universität Bern

Extreme Values of Gaussian Processes and A Heterogeneous Multi Agents Model

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Christoph Manuel Schmid

von Niedermuhlern/BE

Leiter der Arbeit: Prof. Dr. J. Hüsler
Institut für mathematische Statistik
und Versicherungslehre
Universität Bern

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Der Dekan:

G. Jäger

Prof. Dr. G. Jäger

Bern, den 5. Dezember 2002

// Abstract

This PhD thesis consists of two parts that both can be put into a financial context. In the first part we consider the extreme behavior of a certain class of Gaussian processes. We are interested in the probability that a linear combination of Gaussian processes and a deterministic trend exceed a given high boundary. The result of this stochastic problem can be interpreted as the ruin probability of a portfolio of assets. In the second part we introduce the "Heterogeneous Multi Agents Model" which describes the price process of a virtual foreign exchange market. The goals are to reproduce the statistical properties of price time series of real foreign exchange markets and to support the hypothesis of a heterogeneous market.

Revised edition (January 30, 2003)

Contents

Part I

Extreme Values of Gaussian Processes

1 Introduction

For calculating ruin probabilities in insurance or finance, it is important to know the tail behavior of stochastic processes. We consider Gaussian processes, which are widely used in practice. Therefore we calculate the probability that a Gaussian process $Y(t)$ exceeds a certain boundary $u \in \mathbb{R}$ in an interval $T \in \mathbb{R}$

$$P(u) := P\left\{\sup_{t\in T} Y(t) > u\right\}.$$

In general, it is an almost impossible problem to find the distribution of this supremum. Precise formulas are only known for a couple of stationary processes in a finite interval in $\mathbb{R}$. The best we can do, is to calculate formulas for the asymptotic behavior $u \to \infty$ of $P(u)$.

Intuition tells us that $P(u)$ for $u \to \infty$ depends mostly on two factors. First, we expect that the supremum will be achieved in the neighborhood of the point(s) of maximal variance, given that $Var[Y(t)]$ is not constant over T. Secondly, we suppose that local smoothness is of importance in that region. That means, the rougher $Y(t)$, the larger the supremum is expected to be. This is connected with Slepian's inequality.

In this thesis, $Y(t)$ is a linear combination of centered Gaussian processes $X_i(t)$ with variances $d_i t^{2H_i}$ and a trend $-ct^\beta$,

$$Y(t) := \sum_{i=1}^{k} w_i X_i(t) - ct^\beta,$$

where $i, k \in \mathbb{N};\ \beta > H_i;\ 0 < H_i < 1;\ w_i \in \mathbb{R}; c, d_i, H_i, \beta, u \in \mathbb{R}^+$ and $u \to \infty$.

In a financial context we may consider $X(t) := \sum_{i=1}^{k} w_i X_i(t)$ as the portfolio of all risks of eg an insurance company at a given time t, ct^β as fixed income, ie premium payments, up to time t and u as the initial cash reserve of the firm. The biggest liability of the firm after its start of economic activities at time $t = 0$ is then denoted by $\sup_{t>0}(X(t) - ct^\beta)$. Thus

$$P\left\{\sup_{t>0}\left(\sum_{i=1}^{k} w_i X_i(t) - ct^\beta\right) > u\right\}$$

may be interpreted as *ruin probability* of the firm.

Rolski, Schmidli, Schmidt and Teugels [13] give a good introduction into this topic.

2 Preliminaries

We introduce the Gaussian process $Y(t)$, whose tail behavior we are interested in, and the corresponding boundary function $f_u(t)$. Then we reformulate the problem using the time transformation $t \to s$. Since $f_u(s)$ is a non-algebraic function and thus is not easy to deal with, we have to work with approximating functions. We also introduce and examine the Gaussian process $X^{(u)}(s)$.

2.1 The Problem

We are interested in the extreme values of $X(t) - ct^\beta$ with $X(t) := \sum_{i=1}^k w_i X_i(t)$ and where $X_i(t)$ are independent continuous Gaussian processes with $E[X_i(t)] := 0$ and $Var[X_i(t)] := d_i t^{2H_i}$. The constants suffice $0 < H_i < 1;\ \beta > H_i;\ w_i \in \mathbb{R}; c, d_i, H_i, \beta, u \in \mathbb{R}^+$ and $u \to \infty$.

To simplify the notation, we renumber, without loss of generality, the H_i such that $H_1 \geq H_2 \geq \ldots \geq H_k$. Then we introduce the two sets of indices $M := \{1, \ldots, m\}$ with $H_1 = H_2 = \ldots = H_m =: H$ and $J := \{m+1, m+2, \ldots, k\}$ with $H_j < H$ for all $j \in J$.

We see that $X(t)$ is also a Gaussian process with expectation value

$$E[X(t)] = \sum_{i=1}^k w_i E[X_i(t)] = 0$$

and variance

$$\begin{aligned} Var[X(t)] &= \sum_{i=1}^k w_i^2 Var[X_i(t)] + \sum_{i \neq j} Cov[w_i X_i(t), w_j X_j(t)] \\ &= \sum_{i=1}^k w_i^2 d_i t^{2H_i} \\ &= \sum_{i=1}^k W_i t^{2H_i}, \end{aligned}$$

where we defined $W_i := w_i^2 d_i$.

We want to calculate the probability that the stochastic process $X(t) - ct^\beta$ exceeds some boundary u

$$P\left\{\sup_{t>0}(X(t)-ct^{\beta})>u\right\},$$

as $u\to\infty$.

Defining the standardized process $\tilde{X}(t):=\frac{X(t)}{\sqrt{Var[X(t)]}}$ we get

$$\begin{aligned}P\left\{\sup_{t>0}(X(t)-ct^{\beta})>u\right\}&=P\left\{\exists t>0:X(t)>u+ct^{\beta}\right\}\\&=P\left\{\exists t>0:\tilde{X}(t)>\frac{u+ct^{\beta}}{\sqrt{\sum_{i=1}^{k}W_it^{2H_i}}}\right\}.\end{aligned}$$

Solving the original problem is thus equivalent to examine the probability that $\tilde{X}(t)$ exceeds the boundary function

$$f_u(t):=\frac{u+ct^{\beta}}{\sqrt{\sum_{i=1}^{k}W_it^{2H_i}}}$$

at least once

$$P\left\{\sup_{t>0}(X(t)-ct^{\beta})>u\right\}=P\left\{\exists t>0:\tilde{X}(t)>f_u(t)\right\}.$$

2.2 The Boundary Function $f_u(s)$

We define

$$W:=\sum_{i=1}^{m}W_i \tag{1}$$

and rewrite the boundary function $f_u(t)$

$$f_u(t)=\frac{u+ct^{\beta}}{\sqrt{Wt^{2H}}\sqrt{1+\sum_{j\in J}W^{-1}W_jt^{2(H_j-H)}}}=\frac{u+ct^{\beta}}{\sqrt{Wt^{2H}}}(1+\delta(t)),$$

with $\delta(t):=(1+\sum_{j\in J}W^{-1}W_jt^{2(H_j-H)})^{-\frac{1}{2}}-1$.

For reasons that will become clearer later on, we perform a *time transformation*

$$s := W^{\frac{1}{2H}} u^{-\frac{1}{\beta}} t. \tag{2}$$

The boundary function looks in the new time like

$$\begin{aligned} f_u(s) &= \frac{u + c(W^{-\frac{1}{2H}} u^{\frac{1}{\beta}} s)^\beta}{\sqrt{W}(W^{-\frac{1}{2H}} u^{\frac{1}{\beta}} s)^H \sqrt{1 + \sum_{j\in J} W^{-1} W_j (W^{-\frac{1}{2H}} u^{\frac{1}{\beta}} s)^{2(H_j - H)}}} \\ &= \frac{u + cW^{-\frac{\beta}{2H}} u s^\beta}{u^{\frac{H}{\beta}} s^H \sqrt{1 + \sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j - H)} s^{2(H_j - H)}}}. \end{aligned}$$

With $\tilde{c} := cW^{-\frac{\beta}{2H}}$ and

$$\delta_u(s) := \left(1 + \sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j - H)} s^{2(H_j - H)}\right)^{-\frac{1}{2}} - 1$$

we get

$$f_u(s) = u^{1-\frac{H}{\beta}} \frac{1 + \tilde{c} s^\beta}{s^H} (1 + \delta_u(s)).$$

Examining $\delta_u(s)$, we see that $\lim_{s\to 0} \delta_u(s) = -1$ and $\lim_{s\to\infty} \delta_u(s) = 0$. If s is increasing to infinity, $s^{2(H_j - H)}$ strictly decreases to zero, so does the sum over j and hence $\delta_u(s)$ strictly increases from -1 to 0.

We define the function

$$v(s) := \frac{1 + \tilde{c} s^\beta}{s^H} \tag{3}$$

and $s_0 := \operatorname{argmin} v(s)$.

Lemma 2.1 *i) Let $v^{'}(s)$ denote the first derivative of $v(s)$ with respect to s and $v^{''}(s)$ the second one. We get*

$$s_0 = \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{\frac{1}{\beta}}$$

as well as $v'(s) < 0$ *for* $s < s_0$ *and* $v'(s) > 0$ *for* $s > s_0$.

ii) Further we have

$$v(s_0) = \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{-\frac{H}{\beta}} \frac{\beta}{\beta - H} =: A$$

and

$$v''(s_0) = \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{-\frac{H+2}{\beta}} H\beta =: B.$$

Proof: i) The first derivative

$$v'(s) = \frac{\beta \tilde{c} s^{\beta+H-1} - H s^{H-1}(1 + \tilde{c} s^{\beta})}{s^{2H}}$$

is equal to zero if

$$\beta \tilde{c} s_0^{\beta} = H(1 + \tilde{c} s_0^{\beta}),$$

respectively

$$s_0 = \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{\frac{1}{\beta}}.$$

It immediately follows that $v'(s) < 0$ for $s < s_0$ and $v'(s) > 0$ for $s > s_0$.

ii) It is easy to see that

$$v(s_0) = \frac{1 + \tilde{c} s_0^{\beta}}{s_0^{H}} = \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{-\frac{H}{\beta}} \frac{\beta}{\beta - H} =: A.$$

Now we calculate $v''(s_0)$

$$\begin{aligned} v''(s_0) &= \tilde{c}(\beta - H)(\beta - H - 1)s_0^{\beta - H - 2} + (H+1)Hs_0^{-H-2} \\ &= s_0^{-H-2}\left[s_0^{\beta}\tilde{c}(\beta - H)(\beta - H - 1) + (H+1)H\right] \\ &= \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{\frac{-H-2}{\beta}}\left[\left(\frac{H}{\tilde{c}(\beta - H)}\right)\tilde{c}(\beta - H)(\beta - H - 1) + (H+1)H\right] \\ &= \left(\frac{H}{\tilde{c}(\beta - H)}\right)^{\frac{-H-2}{\beta}} H\beta \\ &=: B. \end{aligned}$$

□

Using (3) we get

$$f_u(s) = u^{1-\frac{H}{\beta}} v(s)(1 + \delta_u(s)). \tag{4}$$

Remark 2.1 *The boundary function $f_u(s)$ is continuous, has the limits $\lim_{s\to 0} f_u(s) = \lim_{s\to\infty} f_u(s) = \infty$ and at least one minimum.*

Remark 2.2 *Unfortunately, $f_u(s)$ is a non-algebraic function and thus explicit solutions for the places of the minima of $f_u(s)$ do not exist in the case $k > 1$. Further it is unclear whether the global minimum is unique.*

Because of Remark 2.2 we need an upper and a lower approximation function of $f_u(s)$ to get an idea of the location of its minima. Therefore we define

$$\kappa_u := \sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j - H)}.$$

For $\epsilon(u) > 0$, we introduce the functions

$$f_u^+(s) := u^{1-\frac{H}{\beta}} v(s)(1 + \delta_u(s_0 + \epsilon(u))) \quad (s > 0)$$

and for some small $s_1,\ s_1 < 1,\ s_1 < s_0 + \epsilon(u)$

$$f_u^-(s) := \begin{cases} f_{u,1}^-(s) := u^{1-\frac{H}{\beta}} v(s)(1 + \kappa_u s^{2(H_k - H)})^{-\frac{1}{2}} & (0 < s \le s_1) \\ f_{u,2}^-(s) := u^{1-\frac{H}{\beta}} v(s)(1 + \delta_u(s_1)) & (s_1 < s \le s_0 - \epsilon(u)) \\ f_{u,3}^-(s) := u^{1-\frac{H}{\beta}} v(s)(1 + \delta_u(s_0 - \epsilon(u))) & (s > s_0 - \epsilon(u)). \end{cases}$$

The following thoughts will explain how s_1 is selected.

The expression

$$\begin{aligned}\frac{\partial}{\partial s} f^-_{u,1}(s) =& \frac{u^{1-\frac{H}{\beta}}}{s\left(s^{2H}+\kappa_u s^{2H_k}\right)^{\frac{3}{2}}}\left[\beta\tilde{c}s^{\beta+2H}+\beta\tilde{c}\kappa_u s^{\beta+2H_k}-Hs^{2H}\right.\\ &\left.-H_k\kappa_u s^{2H_k}-\tilde{c}Hs^{\beta+2H}-\tilde{c}H_k\kappa_u s^{\beta+2H_k}\right]\end{aligned}$$

is negative if

$$\tilde{c}(\beta-H)s^{\beta+2H}+\tilde{c}\kappa_u(\beta-H_k)s^{\beta+2H_k}-Hs^{2H}-H_k\kappa_u s^{2H_k}<0,$$

respectively

$$\tilde{c}(\beta-H)s^{\beta+2(H-H_k)}+\tilde{c}\kappa_u(\beta-H_k)s^{\beta}-Hs^{2(H-H_k)}-H_k\kappa_u<0,$$

which is true if $s<s_1$ for some $s_1>0$ small enough.

Hence $f^-_{u,1}(s)$ is strictly decreasing in some interval $(0,s_1)$.

Lemma 2.2 *For $s\in(0,s_0+\epsilon(u))$ and any $\epsilon(u)\geq 0$ we have*

$$f^-_u(s)<f_u(s)<f^+_u(s)$$

and for $s\in[s_0+\epsilon(u),\infty)$

$$f_u(s)\geq f^+_u(s)>f^-_u(s)$$

with equality for $s=s_0+\epsilon(u)$.

Proof: We use the results of Lemma 2.1 and combine them with the strictly increasing property of $1+\delta_u(s)$ and the assumptions made about β and H_i.

□

Lemma 2.3 *The global minimum/minima of $f_u(s)$ is/are in the interval $(0,s_0]$.*

Proof: Using Lemma 2.2 and since $f^+_u(s)$ is strictly increasing for $s>s_0$, we have

$$\min\{f_u(s)|s \geq s_0 + \epsilon(u)\} = f_u^+(s_0 + \epsilon(u)) > \min\{f_u(s)|s < s_0 + \epsilon(u)\},$$

which finishes the proof.

□

Lemma 2.4 *For s_1 suitably chosen, the global minimum/minima of $f_u(s)$ is/are in the interval $[s_1, s_0]$.*

Proof: $f_{u,1}^-(s)$ is strictly decreasing in $(0, s_1)$ and $f_{u,1}^-(s) \to \infty$ for $s \to 0$. $\forall u$ fixed, there exists a $s_1 < s_0$ such that

$$\min_{s \leq s_0} \{f_u(s)\} \leq \min_{s \leq s_1} \left\{f_{u,1}^-(s)\right\} < \min_{s \leq s_1} \{f_u(s)\}.$$

Combining the previous inequalities with Lemma 2.3 finishes the proof.

□

Proposition 2.1 *Defining $s_u^* := \inf\{\text{argmin } f_u(s)\}$, we have $s_u^* \to s_0$ for $u \to \infty$.*

Proof: Lemma 2.4 guarantees that $s_u^* \leq s_0$. It remains to show that if $\forall\, \delta > 0$ there exists a sequence $u(i) \to \infty$ such that $s_{u(i)}^* < s_0 - \delta$ leads to a contradiction. Because s_0 minimizes $v(s)$ and since $s_{u(i)}^* < s_0 - \delta$ we have $v(s_{u(i)}^*)/v(s_0) > 1 + \Delta(\delta)$ for some $\Delta(\delta) > 0$. Further we know that $(1 + \delta_{u(i)}(s_{u(i)}^*))/(1 + \delta_{u(i)}(s_0)) = 1 + o(1)$ since $\delta_{u(i)}(\cdot) = o(1)$, as $u(i) \to \infty$.

Hence

$$\begin{aligned}
f_{u(i)}(s_{u(i)}^*) &= (u(i))^{1-\frac{H}{\beta}} v(s_{u(i)}^*)(1 + \delta_{u(i)}(s_{u(i)}^*)) \\
&= f_{u(i)}(s_0) \frac{v(s_{u(i)}^*)(1 + \delta_{u(i)}(s_{u(i)}^*))}{v(s_0)(1 + \delta_{u(i)}(s_0))} \\
&> f_{u(i)}(s_0)(1 + \Delta(\delta))(1 + o(1)) \\
&> f_{u(i)}(s_0).
\end{aligned}$$

So we get $f_{u(i)}(s_{u(i)}^*) > f_{u(i)}(s_0)$, which can not be true since $s_{u(i)}^* = \inf\{\text{argmin } f_{u(i)}(s)\}$.

□

Proposition 2.2 *The place(s) of the global minimum/minima of $f_u(s)$ are located in the finite interval $[s_u^*, s_0]$.*

Proof: Is a direct consequence of Lemma 2.4 and the definition of s_u^*.

□

Proposition 2.3 *For $u \to \infty$ we get*

$$f_u(s_0) = u^{1-\frac{H}{\beta}} A(1+\delta_u(s_0)) = u^{1-\frac{H}{\beta}} A\left(1 + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right),$$

with A as defined in Lemma 2.1 and

$$f_u(s_u^*) \sim u^{1-\frac{H}{\beta}} A.$$

Proof: Applying Lemma 2.1 we get

$$\begin{aligned}
f_u(s_0) &= u^{1-\frac{H}{\beta}} v(s_0)(1+\delta_u(s_0)) \\
&= u^{1-\frac{H}{\beta}} A\left(1 + \sum_{j\in J} W_j W^{-\frac{H_j}{H}} s_0^{2(H_j-H)} u^{\frac{2}{\beta}(H_j-H)}\right)^{-\frac{1}{2}} \\
&= u^{1-\frac{H}{\beta}} A\left(1 + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right)^{-\frac{1}{2}} \\
&= u^{1-\frac{H}{\beta}} A\left(1 + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right).
\end{aligned}$$

Proposition 2.1 proves then $f_u(s_u^*) \sim u^{1-\frac{H}{\beta}} A$.

□

Proposition 2.4 *For $u \to \infty$ we get*

$$f_u''(s_0) = u^{1-\frac{H}{\beta}} B\left(1 + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right),$$

with B as defined in Lemma 2.1 and

$$f_u''(s_u^*) \sim u^{1-\frac{H}{\beta}} B.$$

Proof: The first and second derivative of $f_u(s) = u^{1-\frac{H}{\beta}} v(s)(1+\delta_u(s))$ with respect to s are given by

$$f_u^{'}(s) = u^{1-\frac{H}{\beta}} v^{'}(s)(1+\delta_u(s)) + u^{1-\frac{H}{\beta}} v(s)(1+\delta_u(s))^{'}$$
$$f_u^{''}(s) = u^{1-\frac{H}{\beta}} v^{''}(s)(1+\delta_u(s)) + 2u^{1-\frac{H}{\beta}} v^{'}(s)(1+\delta_u(s))^{'} + u^{1-\frac{H}{\beta}} v(s)(1+\delta_u(s))^{''}.$$

Calculating $(1+\delta_u(s))^{'}$ yields

$$\begin{aligned}(1+\delta_u(s))^{'} =& -\frac{1}{2}\left(1+\sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)}\right)^{-\frac{3}{2}} \times \\ & \times \sum_{j\in J}\left(2(H_j-H)W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)-1}\right) \\ =& -\frac{1}{2}\left(1+O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right)^{-\frac{3}{2}} O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right) \\ =& O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right),\end{aligned}$$

for $u \to \infty$ and $s_1 < s < \infty$.

Calculating $(1+\delta_u(s))^{''}$ results in

$$\begin{aligned}(1+\delta_u(s))^{''} =& \frac{3}{4}\left(1+\sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)}\right)^{-\frac{5}{2}} \times \\ & \times \left(\sum_{j\in J} 2(H_j-H)W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)-1}\right)^{2} \\ & -\frac{1}{2}\left(1+\sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)}\right)^{-\frac{3}{2}} \times \\ & \times \sum_{j\in J}(2(H_j-H)-1)2(H_j-H)W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)-2} \\ =& O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right).\end{aligned}$$

Inserting the results we have for $u \to \infty$

$$\begin{aligned}
f_u^{''}(s) &= u^{1-\frac{H}{\beta}} \left[v^{''}(s)(1+\delta_u(s)) + v^{'}(s) O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right) + v(s) O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right] \\
&= u^{1-\frac{H}{\beta}} \left[v^{''}(s) + v^{''}(s)\delta_u(s) + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right] \\
&= u^{1-\frac{H}{\beta}} v^{''}(s) \left[1 + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right],
\end{aligned}$$

since $\delta_u(s) = O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)$ and $s \in (s_0 - \hat{\delta}, s_0 + \hat{\delta})$ with $\hat{\delta} > 0$ such that $v^{''}(s) > 0$.

Using Lemma 2.1, we get $f_u^{''}(s_0) = u^{1-\frac{H}{\beta}} B \left(1 + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)\right)$ and with Proposition 2.1, $f_u^{''}(s_u^*) \sim u^{1-\frac{H}{\beta}} B$ because

$$\lim_{u\to\infty} \left[\frac{f_u^{''}(s_u^*)}{u^{1-\frac{H}{\beta}} B}\right] = \frac{v^{''}(s_0)}{B} = 1.$$

□

2.3 The Gaussian Process $X^{(u)}(s)$ and the Interval S_u

For $i = 1, \ldots, k$ we introduce the processes

$$X_i^{(u)}(s) := \frac{X_i(u^{\frac{1}{\beta}} W^{-\frac{1}{2H}} s)}{u^{\frac{H}{\beta}}(1 + \tilde{c}s^{\beta})} \tag{5}$$

and

$$X^{(u)}(s) := \sum_{i=1}^{k} w_i X_i^{(u)}(s) = \frac{X(u^{\frac{1}{\beta}} W^{-\frac{1}{2H}} s)}{u^{\frac{H}{\beta}}(1 + \tilde{c}s^{\beta})}. \tag{6}$$

Lemma 2.5 *For $i \in M \cup J, j \in J$ and $m \in M$, the means are given by*

$$E\big[X_i^{(u)}(s)\big] = 0 \text{ and } E\big[X^{(u)}(s)\big] = 0$$

and the variances by

$$
\begin{aligned}
Var[X_m^{(u)}(s)] &= \frac{d_m}{Wv^2(s)},\\
Var[X_j^{(u)}(s)] &= d_j \frac{s^{2H_j}}{(1+\tilde{c}s^\beta)^2} W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} = o(1) \quad and\\
Var[X^{(u)}(s)] &= \frac{u^{2-\frac{2H}{\beta}}}{f_u^2(s)} = \frac{1}{v^2(s)} + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right),
\end{aligned}
$$

for $u \to \infty$.

Proof: The processes are obviously centered.

Let us calculate the variances for $m \in M$ and $j \in J$,

$$
\begin{aligned}
Var[X_m^{(u)}(s)] &= \frac{Var[X_m(u^{\frac{1}{\beta}} W^{-\frac{1}{2H}} s)]}{u^{\frac{2H}{\beta}}(1+\tilde{c}s^\beta)^2}\\
&= \frac{d_m (u^{\frac{1}{\beta}} W^{-\frac{1}{2H}} s)^{2H}}{u^{\frac{2H}{\beta}}(1+\tilde{c}s^\beta)^2}\\
&= \frac{d_m}{W} \frac{s^{2H}}{(1+\tilde{c}s^\beta)^2}\\
&= \frac{d_m}{Wv^2(s)},\\
Var[X_j^{(u)}(s)] &= \frac{Var[X_j(u^{\frac{1}{\beta}} W^{-\frac{1}{2H}} s)]}{u^{\frac{2H}{\beta}}(1+\tilde{c}s^\beta)^2}\\
&= \frac{d_j (u^{\frac{1}{\beta}} W^{-\frac{1}{2H}} s)^{2H_j}}{u^{\frac{2H}{\beta}}(1+\tilde{c}s^\beta)^2}\\
&= d_j \frac{s^{2H_j}}{(1+\tilde{c}s^\beta)^2} W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)}\\
&= o(1),
\end{aligned}
$$

as $u \to \infty$ and

$$\begin{aligned}
Var[X^{(u)}(s)] &= \sum_{i=1}^{k} w_i^2 Var[X_i^{(u)}(s)] \\
&= \sum_{m} w_m^2 Var[X_m^{(u)}(s)] + \sum_{j\in J} w_j^2 Var[X_j^{(u)}(s)] \\
&= \sum_{m} w_m^2 \frac{d_m}{Wv^2(s)} + \sum_{j\in J} w_j^2 d_j \frac{s^{2H_j}}{(1+\tilde{c}s^{\beta})^2} W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} \\
&= \frac{1}{v^2(s)} \left(1 + \sum_{j\in J} W_j W^{-\frac{H_j}{H}} u^{\frac{2}{\beta}(H_j-H)} s^{2(H_j-H)} \right) \\
&= \frac{1}{v^2(s)(1+\delta_u(s))^2} \\
&= \frac{u^{2-\frac{2H}{\beta}}}{f_u^2(s)}.
\end{aligned}$$

There is also $1\big/\left(v^2(s)(1+\delta_u(s))^2\right) = 1/v^2(s) + O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)$ for all $s>0$, as $u\to\infty$. $\square$

Lemma 2.6 *We assume that* $\tilde{X}_i^{(u)}(s) := \frac{X_i^{(u)}(s)}{\sqrt{Var[X_i^{(u)}(s)]}}$ $(i = 1,\ldots,k)$ *are asymptotically locally stationary,*

$$\lim_{u\to\infty} \frac{E[\tilde{X}_i^{(u)}(s) - \tilde{X}_i^{(u)}(s')]^2}{K_i^2(|s-s'|)} = D_i \tag{7}$$

uniformly for $s, s' \in S_u := [s_u^* - \epsilon(u), s_u^* + \epsilon(u)]$ *with* $D_i > 0$ *and* $\epsilon(u) := u^{\frac{H}{\beta}-1}\log u$. *For* $i \in M$, $K_i^2(\cdot)$ *is regularly varying at 0 with index* $\alpha_i \in (0,2)$.

There exists a function $K^2(\cdot)$ *regularly varying at* 0 *with index* α *such that*

$$\lim_{s\to s'} \frac{\sum_{m\in M} W_m D_m K_m^2(|s-s'|)}{K^2(|s-s'|)} = \tilde{W}.$$

For $i \in J$, $K_i^2(\cdot)$ *is regularly varying at 0 with index* $\alpha_i \in (0,2)$ *such that for* $h < 2\epsilon(u)$: $K_i(h)\big/K(h) = O(1)$, *as* $u \to \infty$.

Then the correlation function of $\tilde{X}^{(u)}(s) := \frac{X^{(u)}(s)}{\sqrt{Var[X^{(u)}(s)]}}$ *has the form*

$$1 - Corr[\tilde{X}^{(u)}(s), \tilde{X}^{(u)}(s')] = \frac{1}{2}E[\tilde{X}^{(u)}(s) - \tilde{X}^{(u)}(s')]^2 \sim \frac{\tilde{W}}{2W} K^2(|s-s'|).$$

Proof: Using the fact that $\tilde{X}^{(u)}(s)$ and $\tilde{X}_i^{(u)}(s)$ are standardized, we get

$$
\begin{aligned}
E\left[\tilde{X}^{(u)}(s)-\tilde{X}^{(u)}(s')\right]^2 &= E\left[\frac{X^{(u)}(s)}{\sqrt{Var[X^{(u)}(s)]}}-\frac{X^{(u)}(s')}{\sqrt{Var[X^{(u)}(s')]}}\right]^2\\
&= E\left[\frac{\sum_{i=1}^k w_i X_i^{(u)}(s)}{\sqrt{Var[X^{(u)}(s)]}}-\frac{\sum_{i=1}^k w_i X_i^{(u)}(s')}{\sqrt{Var[X^{(u)}(s')]}}\right]^2\\
&= E\left[\sum_{i=1}^k w_i^2\left(\frac{X_i^{(u)}(s)}{\sqrt{Var[X^{(u)}(s)]}}-\frac{X_i^{(u)}(s')}{\sqrt{Var[X^{(u)}(s')]}}\right)^2\right]\\
&= E\left[\sum_{i=1}^k w_i^2\left(\sqrt{\frac{Var[X_i^{(u)}(s)]}{Var[X^{(u)}(s)]}}\tilde{X}_i^{(u)}(s)-\sqrt{\frac{Var[X_i^{(u)}(s')]}{Var[X^{(u)}(s')]}}\tilde{X}_i^{(u)}(s')\right)^2\right].
\end{aligned}
$$

To simplify the last expression, we split the sum $\sum_{i=1}^k$ into $\sum_{m\in M}$ and $\sum_{j\in J}$. Using Lemma 2.5, we see that for $m\in M$ and $u\to\infty$

$$
\begin{aligned}
\sqrt{\frac{Var[X_m^{(u)}(s)]}{Var[X^{(u)}(s)]}} &= \sqrt{\frac{d_m}{W}}\frac{1}{v(s)}\Big/\frac{1}{v(s)(1+\delta_u(s))}\\
&= \sqrt{\frac{d_m}{W}}(1+\delta_u(s)).
\end{aligned}
$$

We get

$$
\begin{aligned}
&E\left[\sqrt{\frac{Var[X_m^{(u)}(s)]}{Var[X^{(u)}(s)]}}\tilde{X}_m^{(u)}(s)-\sqrt{\frac{Var[X_m^{(u)}(s')]}{Var[X^{(u)}(s')]}}\tilde{X}_m^{(u)}(s')\right]^2\\
=&E\left[\sqrt{\frac{d_m}{W}}(1+\delta_u(s))\tilde{X}_m^{(u)}(s)-\sqrt{\frac{d_m}{W}}(1+\delta_u(s'))\tilde{X}_m^{(u)}(s')\right]^2\\
=&E\left[\sqrt{\frac{d_m}{W}}(1+\delta_u(s))\left(\tilde{X}_m^{(u)}(s)-\tilde{X}_m^{(u)}(s')\right)+\sqrt{\frac{d_m}{W}}(\delta_u(s)-\delta_u(s'))\tilde{X}_m^{(u)}(s')\right]^2\\
=&\frac{d_m}{W}(1+\delta_u(s))^2E\left[\tilde{X}_m^{(u)}(s)-\tilde{X}_m^{(u)}(s')\right]^2+\frac{d_m}{W}(\delta_u(s)-\delta_u(s'))^2E\left[\tilde{X}_m^{(u)}(s')\right]^2\\
&+2\frac{d_m}{W}(1+\delta_u(s))(\delta_u(s)-\delta_u(s'))E\left[\left(\tilde{X}_m^{(u)}(s)-\tilde{X}_m^{(u)}(s')\right)\tilde{X}_m^{(u)}(s')\right]\\
\sim&\frac{d_m}{W}E\left[\tilde{X}_m^{(u)}(s)-\tilde{X}_m^{(u)}(s')\right]^2,
\end{aligned}
$$

where the last step is possible since, using that $\alpha_m \in (0,2)$ and, for some $\eta \in (s^{'}, s)$ we have $\delta_u(s) - \delta_u(s^{'}) = (s - s^{'})\delta_u^{'}(\eta) = (s - s^{'})O\left(u^{\frac{2}{\beta}(H_{m+1}-H)}\right)$,

$$
\begin{aligned}
&\lim_{u\to\infty} \frac{(\delta_u(s) - \delta_u(s^{'}))^2 E\left[\tilde{X}_m^{(u)}(s)\right]^2}{(1+\delta_u(s))^2 E\left[\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right]^2} \\
&= \lim_{u\to\infty} \frac{(\delta_u(s) - \delta_u(s^{'}))^2 / K_m^2(|s-s^{'}|)}{(1+\delta_u(s))^2 E\left[\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right]^2 / K_m^2(|s-s^{'}|)} \\
&= \lim_{u\to\infty} \frac{(\delta_u(s) - \delta_u(s^{'}))^2}{D_m K_m^2(|s-s^{'}|)} \\
&= \lim_{u\to\infty} \frac{|s-s^{'}|^2 O\left(u^{\frac{4}{\beta}(H_{m+1}-H)}\right)}{D_m |s-s^{'}|^{\alpha_m} L_m(|s-s^{'}|)} \\
&= 0
\end{aligned}
$$

and

$$
\begin{aligned}
&\lim_{u\to\infty} \frac{2(\delta_u(s) - \delta_u(s^{'}))E\left[\left(\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right)\tilde{X}_m^{(u)}(s^{'})\right]}{(1+\delta_u(s))E\left[\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right]^2} \\
&= \lim_{u\to\infty} \frac{2(\delta_u(s) - \delta_u(s^{'}))\left(E\left[\tilde{X}_m^{(u)}(s)\tilde{X}_m^{(u)}(s^{'})\right] - 1\right)}{(1+\delta_u(s))E\left[\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right]^2} \\
&= \lim_{u\to\infty} \frac{2(\delta_u(s) - \delta_u(s^{'}))\left(1 - \frac{1}{2}E\left[\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right]^2 - 1\right)}{(1+\delta_u(s))E\left[\tilde{X}_m^{(u)}(s) - \tilde{X}_m^{(u)}(s^{'})\right]^2} \\
&= \lim_{u\to\infty} \frac{\delta_u(s^{'}) - \delta_u(s)}{1+\delta_u(s)} \\
&= 0.
\end{aligned}
$$

Again, we use Lemma 2.5 and for $j \in J$ we get

$$\begin{aligned}
\sqrt{\frac{Var[X_j^{(u)}(s)]}{Var[X^{(u)}(s)]}} &= \sqrt{d_j}\frac{s^{H_j}}{1+\tilde{c}s^{\beta}}W^{-\frac{H_j}{2H}}u^{\frac{1}{\beta}(H_j-H)}\Big/\frac{1}{v(s)(1+\delta_u(s))} \\
&= \sqrt{d_j}W^{-\frac{H_j}{2H}}u^{\frac{1}{\beta}(H_j-H)}s^{H_j-H}(1+\delta_u(s)) \\
&=: \xi_j(u,s).
\end{aligned}$$

We note that $\xi_j(u,s) = O\left(u^{\frac{1}{\beta}(H_j-H)}\right)$ and for some $\eta \in (s^{'}, s)$

$$\begin{aligned}
\xi_j(u,s) - \xi(u,s^{'}) =& O\left(u^{\frac{1}{\beta}(H_j-H)}\left[s^{2(H_j-H)}(1+\delta_u(s)) - s^{'2(H_j-H)}(1+\delta_u(s^{'}))\right]\right) \\
=& O\left(u^{\frac{1}{\beta}(H_j-H)}\left[(s-s^{'})(s^{2(H_j-H)}(1+\delta_u(s)))^{'}|_{s=\eta}\right]\right) \\
=& O\left(u^{\frac{1}{\beta}(H_j-H)}\left[(s-s^{'})\left(2(H_j-H)\eta^{2(H_j-H)-1}(1+\delta_u(\eta))\right.\right.\right. \\
&\left.\left.\left. +\eta^{2(H_j-H)}(1+\delta_u(s))^{'}|_{s=\eta}\right)\right]\right) \\
=& O\left(u^{\frac{1}{\beta}(H_j-H)}\left(1+O\left(u^{\frac{2}{\beta}(H_j-H)}\right)\right)(s-s^{'})\right) \\
=& O\left(u^{\frac{1}{\beta}(H_j-H)}(s-s^{'})\right).
\end{aligned}$$

We get

$$\begin{aligned}
&E\left[\sqrt{\frac{Var[X_j^{(u)}(s)]}{Var[X^{(u)}(s)]}}\tilde{X}_j^{(u)}(s) - \sqrt{\frac{Var[X_j^{(u)}(s^{'})]}{Var[X^{(u)}(s^{'})]}}\tilde{X}_j^{(u)}(s^{'})\right]^2 \\
=& E\left[\xi_j(u,s)\tilde{X}_j^{(u)}(s) - \xi_j(u,s^{'})\tilde{X}_j^{(u)}(s^{'})\right]^2 \\
=& E\left[\xi_j(u,s)\left(\tilde{X}_j^{(u)}(s) - \tilde{X}_j^{(u)}(s^{'})\right) + \left(\xi_j(u,s)-\xi_j(u,s^{'})\right)\tilde{X}_j^{(u)}(s^{'})\right]^2 \\
=& \xi_j^2(u,s)E\left[\tilde{X}_j^{(u)}(s) - \tilde{X}_j^{(u)}(s^{'})\right]^2 + \left(\xi_j(u,s)-\xi_j(u,s^{'})\right)^2 E\left[\tilde{X}_j^{(u)}(s^{'})\right]^2 \\
&+ 2\xi_j(u,s)\left(\xi_j(u,s)-\xi_j(u,s^{'})\right)E\left[\left(\tilde{X}_j^{(u)}(s) - \tilde{X}_j^{(u)}(s^{'})\right)\tilde{X}_j^{(u)}(s^{'})\right] \\
=& O\left(u^{\frac{2}{\beta}(H_j-H)}|s-s^{'}|^{\alpha_j}L_j(|s-s^{'}|)\right),
\end{aligned}$$

as $u \to \infty$, where we used that

$$\begin{aligned}
&\xi_j^2(u,s)E\left[\tilde{X}_j^{(u)}(s)-\tilde{X}_j^{(u)}(s^{'})\right]^2\\
&\qquad\sim \xi_j^2(u,s)D_jK_j^2(|s-s^{'}|)\\
&\qquad= O\left(u^{\frac{2}{\beta}(H_j-H)}|s-s^{'}|^{\alpha_j}L_j(|s-s^{'}|)\right),
\end{aligned}$$

$$\left(\xi_j(u,s)-\xi_j(u,s^{'})\right)^2 E\left[\tilde{X}_j^{(u)}(s^{'})\right]^2 = O\left(u^{\frac{2}{\beta}(H_j-H)}(s-s^{'})^2\right)$$

and

$$\begin{aligned}
&2\xi_j(u,s)\left(\xi_j(u,s)-\xi_j(u,s^{'})\right)E\left[\left(\tilde{X}_j^{(u)}(s)-\tilde{X}_j^{(u)}(s^{'})\right)\tilde{X}_j^{(u)}(s^{'})\right]\\
&\quad=O\left(u^{\frac{1}{\beta}(H_j-H)}\right)O\left(u^{\frac{2}{\beta}(H_j-H)}(s-s^{'})\right)\left(E\left[\tilde{X}_j^{(u)}(s)\tilde{X}_j^{(u)}(s^{'})\right]-1\right)\\
&\quad=O\left(u^{\frac{2}{\beta}(H_j-H)}(s-s^{'})\right)\left(1-\frac{1}{2}E\left[\tilde{X}_j^{(u)}(s)-\tilde{X}_j^{(u)}(s^{'})\right]^2-1\right)\\
&\quad=O\left(u^{\frac{2}{\beta}(H_j-H)}(s-s^{'})K_j^2(|s-s^{'}|)\right)\\
&\quad=O\left(u^{\frac{2}{\beta}(H_j-H)}(s-s^{'})|s-s^{'}|^{\alpha_j}L_j(|s-s^{'}|)\right)\\
&\quad=O\left(u^{\frac{2}{\beta}(H_j-H)}|s-s^{'}|^{1+\alpha_j}L_j(|s-s^{'}|)\right).
\end{aligned}$$

We note that $\forall s,s^{'}\in S_u:=[s_u^*-\epsilon(u),s_u^*+\epsilon(u)]$

$$|s-s^{'}|\le 2\epsilon(u)\le 2u^{\frac{H}{\beta}-1}\log u = O\left(u^{\frac{H}{\beta}-1}\log u\right).$$

We get

$$\begin{aligned}
E\left[\tilde{X}^{(u)}(s)-\tilde{X}^{(u)}(s^{'})\right]^2 =&(1+o(1))\sum_{m\in M}w_m^2\left(\frac{d_m}{W}E\left[\tilde{X}_m^{(u)}(s)-\tilde{X}_m^{(u)}(s^{'})\right]^2\right)\\
&+\sum_{j\in J}O\left(u^{\frac{4}{\beta}(H_j-H)}|s-s^{'}|^{\alpha_j}L_j(|s-s^{'}|)\right)\\
&\sim\sum_{m\in M}w_m^2\frac{d_m}{W}E\left[\tilde{X}_m^{(u)}(s)-\tilde{X}_m^{(u)}(s^{'})\right]^2\\
&\sim\frac{\tilde{W}}{W}K^2(|s-s^{'}|),
\end{aligned}$$

where we used that for all $j \in J$

$$O\left(u^{\frac{2}{\beta}(H_j-H)}|s-s^{'}|^{\alpha_j}L_j(|s-s^{'}|)\right) = o\left(K^2(|s-s^{'}|)\right)$$

because

$$\lim_{u\to\infty} \frac{u^{\frac{2}{\beta}(H_j-H)}K_j^2(|s-s^{'}|)}{K^2(|s-s^{'}|)} = 0$$

since $H_j < H$ and $K_j(h)/K(h) = O(1)$ for h small.

Finally, the correlation function of $\tilde{X}^{(u)}(s)$ looks for $u \to \infty$, respectively $s \to s^{'}$ like

$$\begin{aligned} 1 - Corr[\tilde{X}^{(u)}(s), \tilde{X}^{(u)}(s^{'})] &= \frac{1}{2}E\left[\tilde{X}^{(u)}(s) - \tilde{X}^{(u)}(s^{'})\right]^2 \\ &\sim \frac{\tilde{W}}{2W}K^2(|s-s^{'}|). \end{aligned}$$

□

Lemma 2.7 *$X^{(u)}(s)$ is bounded in a bounded interval.*

Proof: $X^{(u)}(s)$ has continuous sample paths and is therefore bounded.

□

3 The Main Theorem

In this chapter we formulate the main theorem and the idea of its proof.

3.1 Assumptions and the Main Theorem

Let the Gaussian processes $X_i^{(u)}(s)$ $(i = 1, \dots, k)$ as defined in (5) suffice the following Hölder conditions:

For some $G_i, \gamma_i > 0$ and for all $s, s^{'}$ there is

$$\limsup_{u\to\infty} E\left[X_i^{(u)}(s) - X_i^{(u)}(s^{'})\right]^2 \le G_i|s - s^{'}|^{\gamma_i}. \tag{8}$$

Theorem 3.1 *Let $X_i(t), t > 0, (i = 1, \dots, k)$ be independent centered continuous Gaussian processes with variance $d_i t^{2H_i}$ and $-ct^\beta$ a trend. The constants suffice: $\beta, c, d_i > 0$ and $0 < H_i < 1$ and $H_i < \beta$ as well as $w_i \in \mathbb{R}$. Let A and B as in Lemma 2.1, W as in Eq. (1) and H_α be Pickand's constant. $f_u(s)$ is as in Eq. (4) and $s_u^* := \inf\{\text{argmin } f_u(s)\}$. Assume the conditions of Lemma 2.6 with $0 < \alpha_i < 2$ and Eq. (8). Then, the tail behavior is given by*

$$P\left\{\sup_{t>0}\left(\sum_{i=1}^k w_i X_i(t) - ct^\beta\right) > u\right\} \sim \frac{\left(\sqrt{\frac{\tilde{W}}{W}}A\right)^{\frac{2}{\alpha}} H_\alpha 2^{-\frac{1}{\alpha}} \exp\left\{-\frac{1}{2}f_u^2(s_u^*)\right\}}{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)},$$

as $u \to \infty$.

Remark 3.1 *The definition of Pickand's constant H_α is given in Appendix D.*

Remark 3.2 *In the case $k = 1$, we have $W = W_1 = d_1 w_1^2$, $\tilde{W} = W_1 D_1$, $A = (H/(\tilde{c}(\beta - H)))^{-\frac{H}{\beta}}\beta/(\beta - H)$ and $B = (H/(\tilde{c}(\beta - H)))^{\frac{-H-2}{\beta}}H\beta$. $K^{-1}(u^{\frac{H}{\beta}-1})$ is regularly varying at 0 with index $\frac{2}{\alpha}$ $(0 < \alpha < 2)$ and $f_u(s)$ reduces to $f_u(s) = u^{1-\frac{H}{\beta}}v(s)$. Hence $f_u^2(s_u^*) = u^{2-\frac{2H}{\beta}}v^2(s_0) = u^{2-\frac{2H}{\beta}}A^2$. We note that $\tilde{c} = cW^{-\frac{\beta}{2H}} = c(d_1w_1^2)^{-\frac{\beta}{2H}} = c$, if we choose $d_1 = w_1 = 1$.*

Hence

$$P\left\{\sup_{t>0}\left(X(t) - ct^\beta\right) > u\right\} \sim \frac{\left(\sqrt{D}A\right)^{\frac{2}{\alpha}} H_\alpha 2^{-\frac{1}{\alpha}} \exp\left\{-\frac{1}{2}A^2u^{2-\frac{2H}{\beta}}\right\}}{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)},$$

which is the result of Hüsler and Piterbarg [9].

3.2 Idea of the Proof

Idea of the proof: To simplify the notation we use again $X(t) := \sum_{i=1}^{k} w_i X_i(t)$. With the time transformation (2) as well as (6) we reformulate the original problem

$$\begin{aligned}
P\left\{\sup_{t>0}(X(t) - ct^{\beta}) > u\right\} &= P\left\{\exists t > 0 : X(t) > u + ct^{\beta})\right\} \\
&= P\left\{\exists s > 0 : X(W^{-\frac{1}{2H}} u^{\frac{1}{\beta}} s) > u + c(W^{-\frac{1}{2H}} u^{\frac{1}{\beta}} s)^{\beta})\right\} \\
&= P\left\{\exists s > 0 : X^{(u)}(s) > \frac{u(1+\tilde{c}s^{\beta})}{u^{\frac{H}{\beta}}(1+\tilde{c}s^{\beta})}\right\} \\
&= P\left\{\sup_{s>0} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}.
\end{aligned}$$

Using Proposition 2.2, it is clear that all minima of $f_u(s)$ occur in the interval $S_u := [s_u^* - \epsilon(u), s_u^* + \epsilon(u)]$, where $s_u^* := \inf\{\text{argmin } f_u(s)\}$. We now split $P\left\{\sup_{s>0} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}$ into the probabilities

$$P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}$$

and

$$P\left\{\sup_{s\notin S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}.$$

Then we show that

$$P\left\{\sup_{s\notin S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} = o\left(P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}\right).$$

We need to choose $\epsilon(u)$ so that $\epsilon(u)u^{1-\frac{H}{\beta}} \to \infty$ $(u \to \infty)$. Thus $\epsilon(u) := u^{\frac{H}{\beta}-1} \log u$ will be our choice.

4 Proof of the Main Theorem

4.1 Tail Behavior of $X^{(u)}(s)$ for $s \in S_u$

We derive the probability that $X^{(u)}(s)$ exceeds $f_u(s)$ for $s \in S_u := [s_u^* - \epsilon(u), s_u^* + \epsilon(u)]$ by applying a result of Bräker [5].

Proposition 4.1 *For $u \to \infty$ we have*

$$P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \sim \frac{\left(\sqrt{\frac{\tilde{W}}{W}}A\right)^{\frac{2}{\alpha}} H_\alpha 2^{-\frac{1}{\alpha}} \exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)},$$

with $K(\cdot)$ as defined in Lemma 2.6.

Proof: With Lemma 2.5 we get

$$\begin{aligned} P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} &= P\left\{\exists s \in S_u : \tilde{X}^{(u)}(s) > \frac{u^{1-\frac{H}{\beta}}}{\sqrt{Var[X^{(u)}(s)]}}\right\} \\ &= P\left\{\exists s \in S_u : \tilde{X}^{(u)}(s) > f_u(s)\right\}. \end{aligned}$$

Because we want to apply Theorem E.3 (Bräker), $\tilde{X}^{(u)}(s)$ is not appropriate. We need a process that is independent of u. We thus introduce the Gaussian processes $U_\pm(s)$. The original probability will then be estimated using Theorem E.2 (Slepian's inequality).

First, we recall the following Assertion due to Gnedenko and Korol'uk [8], as well as Geluk and de Haan [7].

Assertion. For any function $R^2(s)$ regularly varying at 0 with index $\alpha \in (0, 2]$, there exists a standardized Gaussian process with correlation function $\rho(s)$ such that $\rho(s) = 1 - R^2(|s|)(1 + o(1)) \geq 0$ as $s \to 0$.

The Assertion and Lemma 2.6 guarantee the existence of two standardized Gaussian processes $U_+(s)$ and $U_-(s)$ so that

$$\lim_{s\to s'}\left[\frac{E[U_\pm(s) - U_\pm(s')]^2}{K^2(|s - s'|)}\right] = \frac{\tilde{W}}{W}(1 \pm \nu),$$

and for $\nu > 0$.

The correlation function is then given by

$$1 - Corr[U_{\pm}(s), U_{\pm}(s^{'})] \sim \frac{\tilde{W}}{2W}(1 \pm \nu)K^2(|s - s^{'}|),$$

for $s \to s^{'}$ and $K^2(|s - s^{'}|)$ as defined in Lemma 2.6.

The processes $\tilde{X}^{(u)}(s), U_+(s)$ and $U_-(s)$ are all standardized and continuous. By construction we have

$$Corr[U_{\pm}(s), U_{\pm}(s^{'})] \lesseqgtr Corr[\tilde{X}^{(u)}(s), \tilde{X}^{(u)}(s^{'})]$$

for $s, s^{'} \in S_u$ and any $\nu > 0$, since $\epsilon(u) \to 0$ for $u \to \infty$.

Applying Theorem E.2 (Slepian's inequality), we get

$$P\left\{\exists s \in S_u : \tilde{X}^{(u)}(s) > f_u(s)\right\} \leq P\left\{\exists s \in S_u : U_+(s) > f_u(s)\right\}$$

and

$$P\left\{\exists s \in S_u : \tilde{X}^{(u)}(s) > f_u(s)\right\} \geq P\left\{\exists s \in S_u : U_-(s) > f_u(s)\right\}.$$

We now calculate the two probabilities

$$g_+(u) := P\left\{\exists s \in S_u : U_+(s) > f_u(s)\right\}$$

and

$$g_-(u) := P\left\{\exists s \in S_u : U_-(s) > f_u(s)\right\}.$$

Then we show that $g_+(u) \sim g_-(u)$ for $u \to \infty$. Hence the probability $P\{\exists s \in S_u : \tilde{X}^{(u)}(s) > f_u(s)\}$ is asymptotically equal to $g_+(u) \sim g_-(u)$.

The Calculation of $g_+(u)$ and $g_-(u)$

We have to verify that $f_u(s)$ satisfies the assumptions $(f1), \ldots, (f5)$ of Theorem E.3 (Bräker) [5], which we want to apply to calculate $g_+(u)$ and $g_-(u)$. We consider $g_+(u)$ first.

(f1) Being an elementary function, $f_u(s)$ is continuous.

(f2) Since $\lim_{u\to\infty} f_u(s) = \infty$ for any $s > 0$, we have $\lim_{u\to\infty}[\inf_{s\in S_u}\{f_u(s)\}] = \infty$.

(f3) With $\psi(x) := \frac{1}{\sqrt{2\pi}x}\exp(-\frac{x^2}{2}) \sim \Psi(x) := 1 - \Phi(x)$ as $x \to \infty$, we have $\forall \epsilon > 0$

$$\int_{S_u} \frac{\psi(\epsilon f_u(s))}{\Delta_u(s)} ds \to 0$$

for $u \to \infty$, because S_u is bounded and (f2) holds.

(f4) Define $G(x) := K^{-1}(1/x)$ $(x > 0)$ and

$$\Delta_u(s) := G\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right) \quad \text{for } s \in S_u.$$

With $g_u(s,\tau) := [f_u(s + \tau\Delta_u(s)) - f_u(s)]f_u(s)$, it is sufficient to show that for $u \to \infty, |\tau| \le \theta, 0 \le \theta < \infty$ and $s \in S_u$

$$|g_u(s,\tau)| \longrightarrow 0.$$

Note that $\lim_{u\to\infty} \tau\Delta_u(s) = \lim_{u\to\infty} \tau G\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right) = \tau G(\infty) = 0.$

For some $\xi \in (s, s + \tau\Delta_u(s))$ we have

$$f_u^{'}(\xi) = \left[\frac{f_u(s + \tau\Delta_u(s)) - f_u(s)}{\tau\Delta_u(s)}\right]$$

and thus

$$g_u(s,\tau) = \tau\Delta_u(s) f_u^{'}(\xi) f_u(s).$$

We now calculate $\Delta_u(s)$ and $f_u^{'}(\xi)$.

$$\begin{aligned}
\Delta_u(s) &:= G\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right) \\
&= K^{-1}\left(1\Big/\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right)\right) \\
&= K^{-1}\left(\sqrt{\frac{W}{\tilde{W}(1+\nu)}} \frac{1}{f_u(s)}\right).
\end{aligned}$$

Propositions C.1 and C.5 guarantee that $K^{-1}(\cdot)$ is regularly varying at 0 with index $\frac{2}{\alpha}$, hence

$$\begin{aligned}\Delta_u(s) &= \left(\frac{W}{\tilde{W}(1+\nu)}\right)^{\frac{2}{\alpha}} \left(\frac{1}{f_u(s)}\right)^{\frac{2}{\alpha}} \tilde{L}\left(\sqrt{\frac{W}{\tilde{W}(1+\nu)}}\frac{1}{f_u(s)}\right)\\ &= O\left(u^{\frac{2}{\alpha}(\frac{H}{\beta}-1)}(1+\delta_u(s))^{-\frac{2}{\alpha}}\tilde{L}\left(\sqrt{\frac{W}{\tilde{W}(1+\nu)}}\frac{1}{f_u(s)}\right)\right),\end{aligned}$$

as $u \to \infty$ and where $\tilde{L}(\cdot)$ is a slowly varying function.

To estimate $f_u^{'}(\xi)$ we use that $f_u^{'}(s_u^*) = 0$ and Proposition 2.4

$$\begin{aligned}f_u^{'}(\xi) =& f_u^{'}(s_u^*) + f_u^{''}(s_u^*)(\xi - s_u^*) + o(\xi - s_u^*)\\ =& O\left(u^{1-\frac{H}{\beta}}(\xi - s_u^*)\right)\\ =& O(\log u),\end{aligned}$$

where we used that $\xi - s_u^* \leq \epsilon(u)$ since $\xi \in S_u$.

Putting together the estimations of $\Delta_u(s)$ and $f_u^{'}(s)$ results in

$$\begin{aligned}|g_u(s,\tau)| =& O(|\tau\Delta_u(s)f_u^{'}(\xi)f_u(s)|)\\ =& O\left(\left|u^{\frac{2}{\alpha}(\frac{H}{\beta}-1)}(1+\delta_u(s))^{-\frac{2}{\alpha}}\tilde{L}\left(\sqrt{\frac{W}{\tilde{W}(1+\nu)}}\frac{1}{f_u(s)}\right)\log u\right.\right.\\ & \left.\left. u^{1-\frac{H}{\beta}}(1+\delta_u(s))\right|\right)\\ =& O\left(u^{(1-\frac{H}{\beta})(1-\frac{2}{\alpha})}(1+\delta_u(s))^{1-\frac{2}{\alpha}}\tilde{L}\left(\sqrt{\frac{W}{\tilde{W}(1+\nu)}}\frac{1}{f_u(s)}\right)\log u\right)\\ =& o(1),\end{aligned}$$

since $\alpha < 2$.

The sequence $g_u(s,\tau)$ converges for $u \to \infty$ to $g(s,\tau) = 0$ for any τ bounded.

(f5) Because $g(s,\tau) = 0$ this condition is obviously satisfied.

Hence we can use Bräker's Theorem since the considered stochastic process $U_+(s)$ is a locally stationary Gaussian process with index $\alpha \in (0,2)$, is independent of the parameter u and the sequence of boundary functions $f_u(s)$ satisfies the conditions (f1), ...,(f5).

We have

$$\lim_{u\to\infty} \frac{1}{\Lambda_u} P\left\{\exists s \in S_u : U_+(s) > f_u(s)\right\} = 1,$$

where $\Lambda_u := \int_{S_u} \lambda_u(s)ds + \lambda_{lu}^- + \lambda_{ru}^+$, with λ_{lu}^- and λ_{ru}^+ as defined in the Appendix.

Since $g(s,\tau) = 0$, Λ_u reduces to $\Lambda_u = \int_{S_u} \lambda_u(s)ds$, where for $s \in S_u$

$$\begin{aligned}\lambda_u(s) &:= \frac{H_\alpha 2^{-\frac{1}{\alpha}} \psi\left(f_u(s)\right)}{G\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right)} \\ &= \frac{H_\alpha 2^{-\frac{1}{\alpha}} \psi(f_u(s))}{K^{-1}\left(1/\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right)\right)}.\end{aligned}$$

The probability is then for $u \to \infty$ given by

$$P\left\{\exists s \in S_u : U_+(s) > f_u(s)\right\} \sim \int_{S_u} \lambda_u(s)ds.$$

We now calculate the integral

$$\begin{aligned}\int_{S_u} \lambda_u(s)ds &= \int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)} \frac{H_\alpha 2^{-\frac{1}{\alpha}} \psi(f_u(s))}{K^{-1}\left(1/\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right)\right)} ds \\ &= H_\alpha 2^{-\frac{1}{\alpha}} \int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)} k(s)\psi(f_u(s))ds,\end{aligned}$$

where

$$k(s) := \frac{1}{K^{-1}\left(1/\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}} f_u(s)\right)\right)}.$$

Lemma 4.1 *For $u \to \infty$ we have*

$$\frac{k(s)}{k(s_0)} \to 1$$

uniform for $s \in S_u$ and

$$k(s_0) \sim \frac{\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}{K^{-1}\left(u^{\frac{H}{\beta}-1}\right)},$$

as $u \to \infty$.

Proof: Because $K^{-1}\left(u^{\frac{H}{\beta}-1}\right)$ is regularly varying at 0 with index $\frac{2}{\alpha}$,

$$\lim_{u\to\infty}\left[\frac{K^{-1}\left(su^{\frac{H}{\beta}-1}\right)}{K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}\right] = s^{\frac{2}{\alpha}}$$

locally uniform on $(0, \infty)$, see [12]. Because $f_u(s)/f_u(s_0)$ is bounded and tends to 1 as $u \to \infty$, we get

$$\begin{aligned}\lim_{u\to\infty}\left[\frac{k(s_0)}{k(s)}\right] &= \lim_{u\to\infty}\left[\frac{K^{-1}\left(1\Big/\sqrt{\frac{\tilde{W}(1+\nu)}{W}}f_u(s)\right)}{K^{-1}\left(1\Big/\sqrt{\frac{\tilde{W}(1+\nu)}{W}}f_u(s_0)\right)}\right]\\ &= \lim_{u\to\infty}\left[\left(\frac{f_u(s_0)}{f_u(s)}\right)^{\frac{2}{\alpha}}\right]\\ &= 1,\end{aligned}$$

since $s \to s_0$ for $u \to \infty$ ($\forall s \in S_u$).

The second statement is true because

$$\lim_{u\to\infty}\left[\frac{k(s_0)K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}\right]$$

$$= \lim_{u\to\infty}\left[\frac{K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{K^{-1}\left(1\Big/\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}f_u(s_0)\right)\right)\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}\right]$$

$$= \lim_{u\to\infty}\left[\frac{K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{K^{-1}\left(u^{\frac{H}{\beta}-1}\Big/\left(A(1+\delta_u(s_0))\sqrt{\frac{\tilde{W}(1+\nu)}{W}}\right)\right)\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}\right]$$

$$= 1,$$

since $K^{-1}(\cdot)$ is regularly varying at 0 with index $\frac{2}{\alpha}$.

□

We note the uniform convergence of $f_u(s)/f_u(s_0) \to 1$, as $u \to \infty$ ($\forall s \in S_u$). Using this statement as well as Proposition 2.3, Lemma 4.1 and $\psi(x) = \frac{1}{\sqrt{2\pi}x}e^{-\frac{x^2}{2}} \sim \Psi(x)$ we get

$$\begin{aligned}
\int_{S_u}\lambda_u(s)ds &= H_\alpha 2^{-\frac{1}{\alpha}}\int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)} k(s)\psi(f_u(s))ds \\
&\sim H_\alpha 2^{-\frac{1}{\alpha}}k(s_0)\int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)}\psi(f_u(s))ds \\
&\sim \frac{H_\alpha 2^{-\frac{1}{\alpha}}\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}{K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}\int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)}\frac{\exp\left(-\frac{1}{2}f_u^2(s)\right)}{\sqrt{2\pi}f_u(s)}ds \\
&\sim \frac{H_\alpha 2^{-\frac{1}{\alpha}}\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}{\sqrt{2\pi}f_u(s_0)K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}\int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)}\exp\left(-\frac{1}{2}f_u^2(s)\right)ds \\
&\sim \frac{H_\alpha 2^{-\frac{1}{\alpha}}\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}}}{\sqrt{2\pi}Au^{1-\frac{H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}\int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)}\exp\left(-\frac{1}{2}f_u^2(s)\right)ds.
\end{aligned}$$

In Proposition 2.1, we showed that $s_u^* \to s_0$ for $u \to \infty$. We now expand the argument in the exponent of the integrand for $s \to s_u^*$

$$
\begin{aligned}
f_u^2(s) = &\left(f_u(s_u^*) + f_u^{'}(s_u^*)(s - s_u^*) + \frac{1}{2} f_u^{''}(s_u^*)(s - s_u^*)^2 + o\left((s - s_u^*)^2\right) \right)^2 \\
= &f_u^2(s_u^*) + f_u(s_u^*) f_u^{''}((s_u^*))(s - s_u^*)^2 + 2 f_u(s_u^*) o\left((s - s_u^*)^2\right) \\
&+ \frac{1}{4}\left(f_u^{''}(s_u^*)\right)^2 (s - s_u^*)^4 + o\left((s - s_u^*)^4\right) \\
= &f_u^2(s_u^*) + f_u(s_u^*) f_u^{''}(s_u^*)(s - s_u^*)^2 (1 + o(1)).
\end{aligned}
$$

The integral is now

$$
\begin{aligned}
&\int_{s_u^* - \epsilon(u)}^{s_u^* + \epsilon(u)} \exp\left(-\frac{1}{2} f_u^2(s)\right) ds \\
&\quad = \int_{s_u^* - \epsilon(u)}^{s_u^* + \epsilon(u)} \exp\left\{-\frac{1}{2}\left(f_u^2(s_u^*) + f_u(s_u^*) f_u^{''}(s_u^*)(s - s_u^*)^2 (1 + o(1))\right)\right\} ds.
\end{aligned}
$$

Remember Propositions 2.3 and 2.4, where we proved $f_u(s_u^*) \sim u^{1-\frac{H}{\beta}} A$ and $f_u^{''}(s_u^*) \sim u^{1-\frac{H}{\beta}} B$.

We change the variable $x := \sqrt{f_u(s_u^*) f_u^{''}(s_u^*)}(s - s_u^*)$ and get $\frac{dx}{ds} = \sqrt{f_u(s_u^*) f_u^{''}(s_u^*)}$. Then $s_u^* - \epsilon(u)$ is replaced by

$$
\begin{aligned}
\sqrt{f_u(s_u^*) f_u^{''}(s_u^*)}(s_u^* - \epsilon(u) - s_u^*) &= -\epsilon(u) \sqrt{f_u(s_u^*) f_u^{''}(s_u^*)} \\
&\sim -(\log u)\sqrt{AB} \\
&\to -\infty \quad (u \to \infty)
\end{aligned}
$$

and $s_u^* + \epsilon(u)$ by

$$
\begin{aligned}
\sqrt{f_u(s_u^*) f_u^{''}(s_u^*)}(s_u^* + \epsilon(u) - s_u^*) &= \epsilon(u) \sqrt{f_u(s_u^*) f_u^{''}(s_u^*)} \\
&= O\left(\log u\right) \\
&\to \infty \quad (u \to \infty).
\end{aligned}
$$

We get

$$\int_{s_u^*-\epsilon(u)}^{s_u^*+\epsilon(u)} \exp\left(-\frac{1}{2}f_u^2(s)\right) ds \sim \frac{\exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{\sqrt{f_u(s_u^*)f_u''(s_u^*)}} \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}x^2(1+o(1))\right) dx$$
$$\sim \frac{\exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{\sqrt{f_u(s_u^*)f_u''(s_u^*)}}\sqrt{2\pi}$$
$$\sim \frac{\sqrt{2\pi}\exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{\sqrt{AB}u^{1-\frac{H}{\beta}}}.$$

For $g_+(u)$ we get the approximation

$$g_+(u) \sim \int_{S_u} \lambda_u(s)ds \sim \frac{H_\alpha 2^{-\frac{1}{\alpha}}\left(\sqrt{\frac{\tilde{W}(1+\nu)}{W}}A\right)^{\frac{2}{\alpha}} \exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}.$$

We derive in the same way

$$g_-(u) \sim \frac{H_\alpha 2^{-\frac{1}{\alpha}}\left(\sqrt{\frac{\tilde{W}(1-\nu)}{W}}A\right)^{\frac{2}{\alpha}} \exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}.$$

Taking the limit $\nu \to 0$, we have

$$P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \sim \frac{\left(\sqrt{\frac{\tilde{W}}{W}}A\right)^{\frac{2}{\alpha}} H_\alpha 2^{-\frac{1}{\alpha}} \exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}.$$

This finishes the proof of Proposition 4.1.

□

4.2 Tail Behavior of $X^{(u)}(s)$ for $s \notin S_u$

We want to calculate the four terms on the right hand side of the following inequality

$$
\begin{aligned}
P\left\{\sup_{s\notin S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq & P\left\{\sup_{0<s\leq s_1} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \\
& + P\left\{\sup_{s_1<s<s_u^*-\epsilon(u)} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \\
& + P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \\
& + P\left\{\sup_{s\geq s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}.
\end{aligned}
$$

The following lemma guarantees that increments of $X^{(u)}(s)$ are finite if the considered time interval is bounded.

Lemma 4.2 *We assume that for $|s-s^{'}| < K < \infty$ the Hölder conditions (8) hold. Then we have*

$$
\limsup_{u\to\infty} E\left[X^{(u)}(s) - X^{(u)}(s^{'})\right]^2 \leq G_K|s-s^{'}|^{\gamma},
$$

where $\gamma := \min\{\gamma_i\} > 0$ and $G_K > 0$ is only depending on K.

Proof: If (8) holds, then for $|s-s^{'}| < K < \infty$ also

$$
\begin{aligned}
\limsup_{u\to\infty} E\left[X^{(u)}(s) - X^{(u)}(s^{'})\right]^2 &= \limsup_{u\to\infty}\left[\sum_{i=1}^{k} w_i^2 E\left[X_i^{(u)}(s) - X_i^{(u)}(s^{'})\right]^2\right] \\
&\leq \sum_{i=1}^{k} w_i^2 G_i |s-s^{'}|^{\gamma_i} \\
&\leq \begin{cases} k_1|s-s^{'}|^{\max\{\gamma_i\}} & (|s-s^{'}|>1) \\ k_2|s-s^{'}|^{\min\{\gamma_i\}} & (|s-s^{'}|\leq 1) \end{cases} \\
&\leq G_K|s-s^{'}|^{\min\{\gamma_i\}}
\end{aligned}
$$

for some $k_1, k_2 > 0$ and some $G_K > 0$ depending on K.

□

Proposition 4.2 *Assume the Hölder conditions (8). For s_1 and $f_{u,1}^-(s)$ as introduced in Chapter 2.2 we get*

$$P\left\{\sup_{0<s\leq s_1} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq C_0 s_1 u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(f_{u,1}^-(s_1)\right),$$

as $u \to \infty$. $\Psi(\cdot)$ denotes the tail of the standard Gauss distribution and the constant C_0 is only depending on G_K and γ.

Proof: For $0 < s \leq s_1$ we have $\min\{f_u(s)\} > \min\{f_{u,1}^-(s)\} = f_{u,1}^-(s_1)$ since $f_{u,1}^-(s)$ is strictly decreasing on $(0, s_1]$.

Because of Lemma 4.2 we can apply Theorem E.1 (Piterbarg) to get

$$P\left\{\sup_{0<s\leq s_1} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq C_0 s_1 u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(\frac{u^{1-\frac{H}{\beta}}}{\sigma_1}\right),$$

for some C_0 only depending on G_K and γ and where, using Lemma 2.5,

$$\begin{aligned}\sigma_1 :=& \max\left\{\sqrt{Var[X^{(u)}(s)]}\right\} = \max\left\{\frac{u^{1-\frac{H}{\beta}}}{f_u(s)}\right\}\\ =& \frac{u^{1-\frac{H}{\beta}}}{\min\{f_u(s)\}} < \frac{u^{1-\frac{H}{\beta}}}{\min\{f_{u,1}^-(s)\}}.\end{aligned}$$

Hence we have

$$\Psi\left(\frac{u^{1-\frac{H}{\beta}}}{\sigma_1}\right) = \Psi\left(\min\{f_u(s)\}\right) < \Psi\left(\min\{f_{u,1}^-(s)\}\right)$$

and

$$P\left\{\sup_{0<s\leq s_1} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq C_0 |s_1| u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(f_{u,1}^-(s_1)\right).$$

□

Proposition 4.3 *Define $\underline{s}_u^* := \inf\{\text{argmin } f_u(s) | s \in (s_1, s_u^* - \epsilon(u))\}$ and assume the Hölder conditions (8). For s_1 as introduced in Chapter 2.2 and for $u \to \infty$ we have*

$$P\left\{\sup_{s_1<s<s_u^*-\epsilon(u)} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq C_1|s_u^*-\epsilon(u)-s_1|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(f_u(\underline{s}_u^*)\right),$$

where the constant C_1 is only depending on γ and G_K.

Proof: Using Lemma 2.5 we see that

$$\sigma_2 := \max_{s\in(s_1,s_u^*-\epsilon(u))}\sqrt{Var[X^{(u)}(s)]} = \frac{u^{1-\frac{H}{\beta}}}{f_u(\underline{s}_u^*)}.$$

Again, Lemma 4.2 provides a Hölder condition so that we can apply Piterbarg's Theorem,

$$\begin{aligned}P\left\{\sup_{s_1<s<s_u^*-\epsilon(u)} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} &\leq C_1|s_u^*-\epsilon(u)-s_1|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(\frac{u^{1-\frac{H}{\beta}}}{\sigma_2})\right)\\ &\leq C_1|s_u^*-\epsilon(u)-s_1|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(f_u(\underline{s}_u^*)\right),\end{aligned}$$

with C_1 only depending on γ and G_K.

□

Proposition 4.4 *For any $s_2 > s_0+\epsilon(u)$ define $\overline{s}_u^* := \inf\{\operatorname{argmin} f_u(s)|s\in(s_u^*+\epsilon(u),s_2)\}$ and assume the Hölder conditions (8). For $u\to\infty$ we have*

$$P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq C_2|s_2-s_u^*-\epsilon(u)|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(f_u(\overline{s}_u^*)\right),$$

where C_2 is only depending on γ and G_K .

Proof: For $s,s^{'}\in(s_u^*+\epsilon(u),s_2)$, Lemma 4.2 does hold. With Theorem E.1 (Piterbarg) we get

$$P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \leq C_2|s_2-s_u^*-\epsilon(u)|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(\frac{u^{1-\frac{H}{\beta}}}{\sigma_3}\right),$$

where C_2 is only depending on γ and G_K with $K=|s_2-s_0-\epsilon(u)|$ and σ_3 is the maximal standard deviation of $X^{(u)}(s)$ in the considered interval,

$$\sigma_3 := \max_{s_u^*+\epsilon(u)<s<s_2} \left\{ \frac{u^{1-\frac{H}{\beta}}}{f_u(s)} \right\} = \frac{u^{1-\frac{H}{\beta}}}{f_u(s_0+\epsilon(u))},$$

where we use Lemma 2.5 and that $f_u(s)$ is strictly increasing for $s > s_0$.

Finally, we get

$$P\left\{ \sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}} \right\} \leq C_2 |s_2 - s_u^* - \epsilon(u)| u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(f_u(\overline{s}_u^*)\right).$$

□

Proposition 4.5 *For $s_2 > s_0 + \epsilon(u)$, we have*

$$\begin{aligned} &P\left\{ \sup_{s\geq s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}} \right\} \\ &= O\left(u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}-1)} \left[u^{\frac{2H}{\beta}-2} + 1 \right] \exp\left(-\frac{1}{2}\tilde{c}^2 s_2^{2(\beta-H)} (1+\delta_u(s_2))^2 u^{2-\frac{2H}{\beta}} \right) \right), \end{aligned}$$

as $u \to \infty$.

Proof: We split the interval $[s_2, \infty)$ into subintervals $I_k = [s_2+k-1, s_2+k)$ $(k = 1, 2, \ldots)$. Lemma 4.2 guarantees that Piterbarg's Theorem can be applied in every subinterval I_k. Since $|s - s^{'}| \leq 1$ for all I_k, we have

$$P\left\{ \sup_{s\in I_k} X^{(u)}(s) > u^{1-\frac{H}{\beta}} \right\} \leq C_4 u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(\frac{u^{1-\frac{H}{\beta}}}{\sigma_k} \right),$$

where C_4 is only depending on G_K and γ but not on k since $|I_k| = K = 1$. However, the maximal standard deviation of $X^{(u)}(s)$ is depending on k,

$$\begin{aligned} \sigma_k &:= \max_{s\in I_k} \left\{ \sqrt{Var[X^{(u)}(s)]} \right\} \\ &= \max_{s\in I_k} \left\{ \frac{u^{1-\frac{H}{\beta}}}{f_u(s)} \right\} = \frac{u^{1-\frac{H}{\beta}}}{f_u(s_2+k-1)}, \end{aligned}$$

where we use Lemma 2.5 and that $f_u(s)$ is strictly increasing for $s > s_0$.

Hence, we get

$$
\begin{aligned}
P\left\{\sup_{s\geq s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} &\leq \sum_{k=1}^{\infty} P\left\{\sup_{s\in I_k} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} \\
&\leq \sum_{k=1}^{\infty} C_4 u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(\frac{u^{1-\frac{H}{\beta}}}{\sigma_k}\right) \\
&\leq \sum_{k=1}^{\infty} C_4 u^{(1-\frac{H}{\beta})\frac{2}{\gamma}} \Psi\left(f_u(s_2+k-1)\right) \\
&\leq \sum_{k=1}^{\infty} \frac{C_4 u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}-1)} \exp\left(-\frac{1}{2} f_u^2(s_2+k-1)\right)}{\sqrt{2\pi} v(s_2+k-1)(1+\delta_u(s_2+k-1))} \\
&\leq \frac{C_4 u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}-1)}}{\sqrt{2\pi} v(s_2)(1+\delta_u(s_2))} \sum_{k=1}^{\infty} \exp\left(-\frac{1}{2} f_u^2(s_2+k-1)\right).
\end{aligned}
$$

With $v^2(s) = (1+2\tilde{c}s^{\beta}+\tilde{c}^2 s^{2\beta})/s^{2H} \geq \tilde{c}^2 s^{2(\beta-H)}$, $1+\delta_u(s_2) \leq 1+\delta_u(s_2+k-1)$ and defining $C(u) := \tilde{c}^2(1+\delta_u(s_2))^2 u^{2-\frac{2H}{\beta}}$ we get

$$
\begin{aligned}
\sum_{k=1}^{\infty} \exp\left(-\frac{1}{2} f_u^2(s_2+k-1)\right) &\leq \sum_{k=1}^{\infty} \exp\left(-\frac{1}{2} u^{2-\frac{2H}{\beta}} v^2(s_2+k-1)(1+\delta_u(s_2))^2\right) \\
&\leq \sum_{k=1}^{\infty} \exp\left(-\frac{1}{2} C(u)(s_2+k-1)^{2(\beta-H)}\right) \\
&\leq \int_{s_2}^{\infty} \exp\left(-\frac{1}{2} C(u) x^{2(\beta-H)}\right) dx + \exp\left(-\frac{1}{2} C(u) s_2^{2(\beta-H)}\right),
\end{aligned}
$$

where the last inequality holds because $s_2+k-1 \geq s$ for $s \in I_{k-1}$ $(k=2,3,\ldots)$.

We need the following integral estimation.

Lemma 4.3 *For $l, \alpha > 0$ and any $S > 0$ we have*

$$
\int_S^{\infty} e^{-lx^{\alpha}} dx \leq \frac{e^{-lS^{\alpha}} (lS^{\alpha})^{\frac{1}{\alpha}-1}}{\alpha l^{\frac{1}{\alpha}} \left[1-(\frac{1}{\alpha}-1)\frac{1}{lS^{\alpha}}\right]}
$$

and

$$
\int_S^{\infty} e^{-lx^{\alpha}} dx \sim \frac{1}{\alpha l^{\frac{1}{\alpha}}} e^{-lS^{\alpha}} (lS^{\alpha})^{\frac{1}{\alpha}-1},
$$

for $l \to \infty$.

Proof: We use the transformation $y := lx^\alpha$, hence $x = \left(\frac{y}{l}\right)^{\frac{1}{\alpha}}$ with $\frac{dx}{dy} = \frac{1}{\alpha} l^{-\frac{1}{\alpha}} y^{\frac{1}{\alpha}-1}$. Integrating by parts gives

$$\begin{aligned}
\int_S^\infty e^{-lx^\alpha} dx &= \frac{1}{\alpha l^{\frac{1}{\alpha}}} \int_{lS^\alpha}^\infty e^{-y} y^{\frac{1}{\alpha}-1} dy \\
&= \frac{1}{\alpha l^{\frac{1}{\alpha}}} \left[e^{-lS^\alpha} (lS^\alpha)^{\frac{1}{\alpha}-1} + \left(\frac{1}{\alpha} - 1\right) \int_{lS^\alpha}^\infty e^{-y} y^{\frac{1}{\alpha}-2} dy \right] \\
&\leq \frac{1}{\alpha l^{\frac{1}{\alpha}}} \left[e^{-lS^\alpha} (lS^\alpha)^{\frac{1}{\alpha}-1} + \left(\frac{1}{\alpha} - 1\right) \frac{1}{lS^\alpha} \int_{lS^\alpha}^\infty e^{-y} y^{\frac{1}{\alpha}-1} dy \right]
\end{aligned}$$

by applying $\int_x^\infty e^{-y} y^{\beta-1} dy \leq \frac{1}{x} \int_x^\infty e^{-y} y^\beta dy$.

Hence we get

$$\int_{lS^\alpha}^\infty e^{-y} y^{\frac{1}{\alpha}-1} dy \left[1 - \left(\frac{1}{\alpha} - 1\right) \frac{1}{lS^\alpha} \right] \leq e^{-lS^\alpha} (lS^\alpha)^{\frac{1}{\alpha}-1}$$

and

$$\int_S^\infty e^{-l^\alpha} dx \leq \frac{e^{-lS^\alpha} (lS^\alpha)^{\frac{1}{\alpha}-1}}{\alpha l^{\frac{1}{\alpha}} \left[1 - (\frac{1}{\alpha} - 1) \frac{1}{lS^\alpha}\right]}.$$

Because of

$$\lim_{l\to\infty} \left[\frac{1}{1 - \left(\frac{1}{\alpha} - 1\right) \frac{1}{lS^\alpha}} \right] = 1$$

we have

$$\frac{e^{-lS^\alpha} (lS^\alpha)^{\frac{1}{\alpha}-1}}{\alpha l^{\frac{1}{\alpha}} \left[1 - \left(\frac{1}{\alpha} - 1\right) \frac{1}{lS^\alpha}\right]} \sim \frac{e^{-lS^\alpha} (lS^\alpha)^{\frac{1}{\alpha}-1}}{\alpha l^{\frac{1}{\alpha}}},$$

as $l \to \infty$.

□

We return to the proof of Proposition 4.5 and use Lemma 4.3, which we can apply since $C(u) \to \infty$ for $u \to \infty$.

$$\int_{s_2}^{\infty} \exp\left(-\frac{1}{2}C(u)x^{2(\beta-H)}\right) dx \sim \frac{\left(\frac{C(u)}{2}s_2^{2(\beta-H)}\right)^{\frac{1}{2(\beta-H)}-1}}{2(\beta-H)\left(\frac{C(u)}{2}\right)^{\frac{1}{2(\beta-H)}}} \exp\left(-\frac{1}{2}C(u)s_2^{2(\beta-H)}\right)$$
$$= O\left(C^{-1}(u)\exp\left(-\frac{1}{2}C(u)s_2^{2(\beta-H)}\right)\right).$$

Putting the various terms together, we get

$$\begin{aligned}\sum_{k=1}^{\infty} \exp\left(-\frac{1}{2}f_u^2(s_2+k-1)\right) \leq & O\left(C^{-1}(u)\exp\left(-\frac{1}{2}C(u)s_2^{2(\beta-H)}\right)\right) \\ & + \exp\left(-\frac{1}{2}C(u)s_2^{2(\beta-H)}\right) \\ = & O\left(\left[u^{\frac{2H}{\beta}-2}+1\right]\exp\left(-\frac{1}{2}C(u)s_2^{2(\beta-H)}\right)\right)\end{aligned}$$

and

$$P\left\{\sup_{s\geq s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\} = O\left(u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}-1)}\left[u^{\frac{2H}{\beta}-2}+1\right]\exp\left(-\frac{1}{2}C(u)s_2^{2(\beta-H)}\right)\right).$$

$\square$

4.3 Dominating Probability

It is clear that

$$\begin{aligned}P\left\{\sup_{t>0}(X(t)-ct^{\beta})>u\right\} \leq & P\left\{\sup_{0<s\leq s_1} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}+P\left\{\sup_{s_1<s<s_u^*-\epsilon(u)} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}\\ &+P\left\{\sup_{s\in S_u} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}+P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}\\ &+P\left\{\sup_{s\geq s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}.\end{aligned}$$

To get

$$P\left\{\sup_{t>0}(X(t)-ct^{\beta})>u\right\}\sim P\left\{\sup_{s\in S_u} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\},$$

it suffices to show that

$$P\left\{\sup_{0<s\leq s_1} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\} = o\left(P\left\{\sup_{s\in S_u} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}\right), \tag{9}$$

$$P\left\{\sup_{s_1<s<s_u^*-\epsilon(u)} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\} = o\left(P\left\{\sup_{s\in S_u} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}\right), \tag{10}$$

$$P\left\{\sup_{s\geq s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\} = o\left(P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}\right) \tag{11}$$

and

$$P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}=o\left(P\left\{\sup_{s\in S_u} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}\right). \tag{12}$$

Lemma 4.4 *For $\underline{s}_u^*$ as defined in Proposition 4.3 and any $\chi>0$ we have for $u\to\infty$*

$$\exp\left(-\frac{1}{2}\left(f_u^2(\underline{s}_u^*)-f_u^2(s_u^*)\right)\right)=o(u^{-\chi}).$$

Proof: By definition of s_u^* and $\underline{s}_u^*$ we have $f_u(s_u^*) < f_u(\underline{s}_u^*)$ and thus $v(s_u^*)(1+\delta_u(s_u^*)) < v(\underline{s}_u^*)(1+\delta_u(\underline{s}_u^*))$.

If $\underline{s}_u^* \not\to s_u^*$ for $u \to \infty$, we have

$$\begin{aligned} f_u^2(\underline{s}_u^*) - f_u^2(s_u^*) &= u^{2-\frac{2H}{\beta}} \left(v^2(\underline{s}_u^*)(1+\delta_u(\underline{s}_u^*))^2 - v^2(s_u^*)(1+\delta_u(s_u^*))^2\right) \\ &\geq const \cdot u^{2-\frac{2H}{\beta}} \end{aligned}$$

and thus $f_u^2(\underline{s}_u^*) - f_u^2(s_u^*) > 2\chi \log u$ for any $\chi > 0$ and u large.

If $\underline{s}_u^* \to s_u^*$ for $u \to \infty$, we use Proposition 2.4 and that $f_u(\underline{s}_u^*) \geq f_u(s_u^*) + (1/2 - \tilde{\delta}(u)) f_u''(s_u^*)(\underline{s}_u^* - s_u^*)^2$ for some $\tilde{\delta}(u) > 0$ such that $\tilde{\delta}(u) = o(1)$, as $u \to \infty$. Hence

$$\begin{aligned} f_u^2(\underline{s}_u^*) - f_u^2(s_u^*) &= (f_u(\underline{s}_u^*) + f_u(s_u^*))(f_u(\underline{s}_u^*) - f_u(s_u^*)) \\ &\geq 2f_u(s_u^*)(f_u(\underline{s}_u^*) - f_u(s_u^*)) \\ &\geq 2f_u(s_u^*)\left(\frac{1}{2} - \tilde{\delta}(u)\right) u^{1-\frac{H}{\beta}} B(1+o(1))(\underline{s}_u^* - s_u^*)^2 \\ &\geq 2f_u(s_u^*)\left(\frac{1}{2} - \tilde{\delta}(u)\right) (\underline{s}_u^* - s_u^*)^2 u^{1-\frac{H}{\beta}} \tilde{B} \\ &\geq 2f_u(s_u^*)\left(\frac{1}{2} - \tilde{\delta}(u)\right) \epsilon^2(u) u^{1-\frac{H}{\beta}} \tilde{B} \\ &\geq \tilde{B} v(s_u^*)(1+\delta_u(s_u^*))(\log u)^2 \\ &> 2\chi \log u \end{aligned}$$

for any $\chi > 0$, some $\tilde{B} < B(1+o(1))$ and u large. We also used that $s_u^* - \underline{s}_u^* > \epsilon(u)$ because of $\underline{s}_u^* < s_u^* - \epsilon(u)$.

We get

$$\exp\left(-\frac{1}{2}\left(f_u^2(\underline{s}_u^*) - f_u^2(s_u^*)\right)\right) < \exp\left(-\chi \log u\right) = u^{-\chi},$$

for any $\chi > 0$ and u large.

$\square$

We now prove (9) using Propositions 4.1 and 4.2

$$
\begin{aligned}
&\lim_{u\to\infty}\left[\frac{P\left\{\sup_{0<s\le s_1} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}}{P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}}\right]\\
&=\lim_{u\to\infty}\left[C_0|s_1|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(f_{u,1}^{-}(s_1)\right)\frac{A\sqrt{AB}u^{2-\frac{H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{\left(\sqrt{\frac{\tilde{W}}{W}}A\right)^{\frac{2}{\alpha}}H_\alpha 2^{-\frac{1}{\alpha}}\exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}\right]\\
&=\lim_{u\to\infty}\left[\frac{u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{f_{u,1}^{-}(s_1)}\exp\left(-\frac{1}{2}\left(\left(f_{u,1}^{-}(s_1)\right)^2-f_u^2(s_u^*)\right)\right)\right]\\
&=\lim_{u\to\infty}\left[\frac{\tilde{L}(u^{\frac{H}{\beta}-1})}{1+o(1)}u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}+1-\frac{2}{\alpha})}\exp\left(-\frac{1}{2}\left(\left(f_{u,1}^{-}(s_1)\right)^2-f_u^2(s_u^*)\right)\right)\right]\\
&=\lim_{u\to\infty}\left[\frac{\tilde{L}(u^{\frac{H}{\beta}-1})}{1+o(1)}e^{\left(1-\frac{H}{\beta}\right)\left(\frac{2}{\gamma}+1-\frac{2}{\alpha}\right)\log u}\exp\left(-\frac{1}{2}\left(\left(f_{u,1}^{-}(s_1)\right)^2-f_u^2(s_u^*)\right)\right)\right]\\
&=0,
\end{aligned}
$$

because

$$
\begin{aligned}
\left(f_{u,1}^{-}(s_1)\right)^2-f_u^2(s_u^*) &= u^{2-\frac{2H}{\beta}}\left[v^2(s_1)(1+o(1))-v^2(s_u^*)(1+o(1))\right]\\
&= u^{2-\frac{2H}{\beta}}\hat{\epsilon},
\end{aligned}
$$

as $u\to\infty$ since $v^2(s_u^*)\to A^2$ and $v^2(s_1)>A^2+\hat{\epsilon}$ for some $\hat{\epsilon}>0$ and $s_1>0$ small enough, keeping in mind that $v(s)\to\infty$ for $s\to 0$.

With Propositions 4.1 and 4.3 we prove (10)

$$
\begin{aligned}
&\lim_{u\to\infty}\left[\frac{P\left\{\sup_{s_1<s<s_u^*-\epsilon(u)} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}}{P\left\{\sup_{s\in S_u} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}}\right]\\
&=\lim_{u\to\infty}\left[C_1|s_u^*-\epsilon(u)-s_1|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi(f_u(\underline{s}_u^*))\frac{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{\left(\sqrt{\frac{\tilde{W}}{W}}A\right)^{\frac{2}{\alpha}}H_\alpha 2^{-\frac{1}{\alpha}}\exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}\right]\\
&=\lim_{u\to\infty}\left[|s_u^*-\epsilon(u)-s_1|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}f_u^{-1}(\underline{s}_u^*)u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)\times\right.\\
&\quad\left.\times\exp\left(-\frac{1}{2}\left(f_u^2(\underline{s}_u^*)-f_u^2(s_u^*)\right)\right)\right]\\
&=\lim_{u\to\infty}\left[|s_u^*-\epsilon(u)-s_1|\exp\left(\left(1-\frac{H}{\beta}\right)\left(\frac{2}{\gamma}+1-\frac{2}{\alpha}\right)\log u-\chi\log u\right)\right]\\
&=0,
\end{aligned}
$$

where we used Lemma 4.4 and χ may be chosen such that $\chi>\left(1-\frac{H}{\beta}\right)\left(\frac{2}{\gamma}+1-\frac{2}{\alpha}\right)$.

To prove (11) we use Propositions 4.4 and 4.5

$$
\begin{aligned}
&\lim_{u\to\infty}\left[\frac{P\left\{\sup_{s\ge s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}}{P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s)>u^{1-\frac{H}{\beta}}\right\}}\right]\\
&=\lim_{u\to\infty}\left[\frac{u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}-1)}\left[u^{\frac{2H}{\beta}-2}+1\right]\exp\left(-\frac{1}{2}\tilde{c}^2 s_2^{2(\beta-H)}(1+\delta_u(s_2))^2u^{2-\frac{2H}{\beta}}\right)}{C_2|s_2-s_u^*-\epsilon(u)|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(f_u\overline{s}_u^*\right)}\right]\\
&=\lim_{u\to\infty}\left[u^{(1-\frac{H}{\beta})(\frac{2}{\gamma}-1)}\left[u^{\frac{2H}{\beta}-2}+1\right]\exp\left(-\frac{1}{2}\tilde{c}^2 s_2^{2(\beta-H)}(1+\delta_u(s_2))^2u^{2-\frac{2H}{\beta}}\right)\times\right.\\
&\quad\left.\times\frac{u^{1-\frac{H}{\beta}}v(\overline{s}_u^*)(1+\delta_u(\overline{s}_u^*))}{u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\exp\left(-\frac{1}{2}f_u^2(\overline{s}_u^*)\right)}\right]\\
&=\lim_{u\to\infty}\left[\exp\left(-\frac{1}{2}u^{2-\frac{2H}{\beta}}\left(\tilde{c}^2 s_2^{2(\beta-H)}(1+\delta_u(s_2))^2-v^2(\overline{s}_u^*)(1+\delta_u(\overline{s}_u^*))^2\right)\right)\right]\\
&=0,
\end{aligned}
$$

if we choose s_2 large enough and using that $\overline{s}_u^*\to s_0$ for $u\to\infty$ because of Propositions 2.1 and 2.2.

To prove (12) we use Propositions 4.1 and 4.4

$$
\begin{aligned}
&\lim_{u\to\infty}\left[\frac{P\left\{\sup_{s_u^*+\epsilon(u)<s<s_2} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}}{P\left\{\sup_{s\in S_u} X^{(u)}(s) > u^{1-\frac{H}{\beta}}\right\}}\right]\\
&=\lim_{u\to\infty}\left[C_2|s_2-s_u^*-\epsilon(u)|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}\Psi\left(f_u(\overline{s}_u^*)\right)\frac{A\sqrt{AB}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)}{\left(\sqrt{\frac{\tilde{W}}{W}}A\right)^{\frac{2}{\alpha}}H_\alpha 2^{-\frac{1}{\alpha}}\exp\left(-\frac{1}{2}f_u^2(s_u^*)\right)}\right]\\
&=\lim_{u\to\infty}\Bigg[\ |s_2-s_u^*-\epsilon(u)|u^{(1-\frac{H}{\beta})\frac{2}{\gamma}}u^{2-\frac{2H}{\beta}}K^{-1}\left(u^{\frac{H}{\beta}-1}\right)f_u^{-1}(\overline{s}_u^*)\times\\
&\quad\times\exp\left(-\frac{1}{2}\left(f_u^2(\overline{s}_u^*)-f_u^2(s_u^*)\right)\right)\Bigg]\\
&=\lim_{u\to\infty}\left[\frac{\tilde{L}\left(u^{\frac{H}{\beta}-1}\right)}{1+o(1)}\exp\left(\left(1-\frac{H}{\beta}\right)\left(\frac{2}{\gamma}+1-\frac{2}{\alpha}\right)\log u-\frac{1}{2}\left(f_u^2(\overline{s}_u^*))-f_u^2(s_u^*)\right)\right)\right]\\
&=0,
\end{aligned}
$$

where we use that $f_u^2(\overline{s}_u^*)-f_u^2(s_u^*)\gg\chi\log u$ for u large, as can be derived analogously to the case $\underline{s}_u^*\to s_u^*$ in the proof of Lemma 4.4,

$$
\begin{aligned}
f_u^2(\overline{s}_u^*)-f_u^2(s_u^*) &= (f_u(\overline{s}_u^*)+f_u(s_u^*))(f_u(\overline{s}_u^*)-f_u(s_u^*))\\
&\geq (f_u(\overline{s}_u^*)+f_u(s_u^*))\frac{1}{2}u^{1-\frac{H}{\beta}}B(1+o(1)))(\overline{s}_u^*-s_u^*)^2\\
&\geq (f_u(\overline{s}_u^*)+f_u(s_u^*))\frac{\tilde{B}}{2}u^{1-\frac{H}{\beta}}\epsilon^2(u)\\
&\geq 2f_u(s_u^*)\frac{\tilde{B}}{2}u^{1-\frac{H}{\beta}}\epsilon^2(u)\\
&\geq A\left(1+o(1)\right)\left(1+\delta_u(s_u^*)\right)\tilde{B}(\log u)^2\\
&\geq \tilde{A}\tilde{B}(\log u)^2\\
&\gg \chi\log u
\end{aligned}
$$

for some $\tilde{A}<A(1+o(1))(1+\delta_u(s_u^*))$ and $\tilde{B}<B(1+o(1))$. We also used that $\overline{s}_u^*-s_u^*>\epsilon(u)$.

So the proof of the Main Theorem is complete.

□

A Gaussian Processes

Definition A.1 *A random vector* $\mathbf{X} = (X_1, \ldots, X_n) \in \mathbb{R}^n$ *has a* multivariate Gaussian distribution *if and only if* $\sum_{i=1}^{n} \alpha_i X_i$ *has a univariate Gaussian distribution for all* $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$.

Remark A.1 *This distribution is determined by a vector* $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$ *and a non-negative definite* $n \times n$ *matrix* $\boldsymbol{\Sigma}$ *of covariances. We use the notation* $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Proposition A.1 X *has the distribution* $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ *if and only if it has the characteristic function*

$$E[\exp(i\mathbf{t}' \cdot \mathbf{X})] = \exp\left(i\mathbf{t}' \cdot \boldsymbol{\mu} - \frac{1}{2}\mathbf{t}'\boldsymbol{\Sigma}\mathbf{t}\right) \quad (\mathbf{t} \in \mathbb{R}^n).$$

If $\boldsymbol{\Sigma}$ *is positive definite (so non-singular),* $\mathbf{X}$ *has the density*

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{det\boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right).$$

Definition A.2 *A stochastic process* $X(t)$ $(t \in T)$*, where* T *is an arbitrary set of parameters, is called a* Gaussian process *if for all* $t_1, \ldots, t_n \in T$ *and any* $n \geq 1$ *the random vector* $(X(t_1), \ldots, X(t_n))$ *has a multivariate Gaussian distribution.*

Remark A.2 *A Gaussian process is uniquely defined by its mean function* $E[X(t)] = \mu(t)$ *and a non-singular covariance function* $Cov[X(t), X(t')]$.

If $\mu(t) = 0$, we call $X(t)$ centered and if we even have $Var[X(t)] = 1$, then $X(t)$ is called standardized Gaussian process.

B Landau Symbols

The *Landau symbols* $O(\cdot)$ and $o(\cdot)$ are used to describe the behavior of two functions $f(x)$ and $g(x)$ relative to each other at some place $x_0 \in \mathbb{R} \cup \pm\infty$.

They are defined in the following way:

For $A \in \mathbb{R} \setminus \{0\}$ and $x \to x_0$ we have

$$f(x) = O(g(x)) \quad \text{if} \quad \lim_{x \to x_0} \left| \frac{f(x)}{g(x)} \right| \leq A$$

and

$$f(x) = o(g(x)) \quad \text{if} \quad \lim_{x \to x_0} \frac{f(x)}{g(x)} = 0.$$

$f(x) \sim g(x)$ is a shorthand notation for

$$\lim_{x \to x_0} \frac{f(x)}{g(x)} = 1.$$

Given $f(x) \sim g(x)$ and $g(x) \sim h(x)$, then also $f(x) \sim h(x)$.

C Regular Varying Functions

We give a brief introduction into the theory of regularly varying functions. A full theory is given in Bingham, Goldie and Teugels [2].

Definition C.1 *A function* $K(x) : \mathbb{R}_+ \to \mathbb{R}_+$ varies regularly at 0 *with index* $\alpha \in \mathbb{R}$, *if* $\forall x \in \mathbb{R}_+$

$$\lim_{t \to 0} \frac{K(xt)}{K(t)} = x^{\alpha}.$$

If $\alpha = 0$, $K(\cdot)$ *is called* slowly varying at 0.

Proposition C.1 *If* $K(x)$ *is regularly varying at 0 with index* α, *then for* $c \in \mathbb{R} \setminus \{0\}$ $K^c(x)$ *is regularly varying at 0 with index* $c\alpha$.

Proof:

$$\lim_{t \to 0} \frac{K^c(xt)}{K^c(t)} = \lim_{t \to 0} \left[\frac{K(xt)}{K(t)} \right]^c = [x^{\alpha}]^c = x^{c\alpha}$$

□

Proposition C.2 *Let* $K(x)$ *be regularly varying at 0 with index* α. *Then*

$$K(x) = x^{\alpha} L(x),$$

for some slowly varying function $L(x)$.

Proof: We have to check whether $K(x)/x^{\alpha}$ is slowly varying,

$$\lim_{t \to 0} \left[\frac{K(xt)}{(xt)^{\alpha}} \Big/ \frac{K(t)}{t^{\alpha}} \right] = \frac{1}{x^{\alpha}} \lim_{t \to 0} \frac{K(xt)}{K(t)} = 1.$$

□

Proposition C.3 *If* $K(x)$ *is regularly varying at 0 with index* α, *it is equivalent to say that* $K(1/x)$ *is regularly varying at* ∞ *with index* $-\alpha$.

Proof: We have

$$\lim_{t \to \infty} \left[\frac{K\left(\frac{1}{tx}\right)}{K\left(\frac{1}{t}\right)} \right] = \lim_{s \to 0} \left[\frac{K(\frac{s}{x})}{K(s)} \right] = \left(\frac{1}{x} \right)^{\alpha} = x^{-\alpha}.$$

□

Proposition C.4 *If $K(x)$ is regularly varying at 0 with index $\alpha \neq 0$, we have*

$$\lim_{x \to 0} K(x) = \begin{cases} 0 & (\alpha > 0), \\ \infty & (\alpha < 0). \end{cases}$$

Proof: See Bingham [3].

□

Proposition C.5 *Let $K(x)$ be defined and locally bounded on $(0, x_0]$ and tend to 0 as $x \to 0$. Its generalized inverse $K^{-1}(x) := \inf\{y \in (0, x_0] : K(y) > x\}$ is defined on $(0, K(x_0)]$ and is monotone decreasing to 0.*

Be $K(x)$ regularly varying at 0 with index $\alpha > 0$. Then $K^{-1}(x)$ is regularly varying at 0 with index $1/\alpha$ and has the representation

$$K^{-1}(x) = x^{\frac{1}{\alpha}} \tilde{L}(x),$$

where $\tilde{L}(x)$ is slowly varying at 0.

Proof: See Bingham [4].

□

D Pickand's Constant H_α

The constant H_α may be defined in several ways. We use the definition

$$H_\alpha := \lim_{T\to\infty} \frac{1}{T} E\left(\exp\left[\max_{0\le t\le T} \chi(t)\right]\right),$$

where $\chi(t)$, $t \ge 0$, is a fractional Brownian motion with drift $E[\chi(t)] = -t^\alpha$ and covariance function $Cov[\chi(s), \chi(t)] = t^\alpha + s^\alpha - |t-s|^\alpha$.

We note that Bräker [6] defined H_α with respect to a fractional Brownian motion with variance s^α, whereas in our definition the variance is given by $2s^\alpha$. Therefore we have

$$H_\alpha(\text{Bräker}) = H_\alpha 2^{-\frac{1}{\alpha}}.$$

In general, it is not known how to calculate H_α. The exceptions are $H_{\alpha=1} = 1$ and $H_{\alpha=2} = 1/\sqrt{\pi}$. Some numerical bounds are given in a paper by Shao [14] and simulated values are listed in Piterbarg and Romanova [11].

E Various Theorems

We present the theorems used to prove the Main Theorem 3.1. For the sake of simplicity we sometimes modify the original notation and restrict ourselves to the statements necessary for our purposes.

Theorem E.1 (Piterbarg) *Let $X(t)$, $t \in T \subset \mathbb{R}$, be a Gaussian process with zero mean, variance $\sigma^2(t)$ and continuous trajectories satisfying*

$$E[X(t) - X(t')]^2 \leq G|t - t'|^\gamma$$

for some $\gamma, G > 0$. Let $|T| > 0$. Then there exists a constant C depending only on G and γ such that for any $S \subset T$,

$$P\left\{\max_{t \in S} |X(t)| > u\right\} \leq C|T|u^{\frac{2}{\gamma}} \Psi\left(\frac{u}{\sigma(S)}\right),$$

where $\sigma(S) := \sup_{t \in S} \sigma(t)$.

Proof: Piterbarg proved the theorem for a Gaussian field in $\mathbb{R}^n$ $(n \in \mathbb{N})$ [10].

□

Theorem E.2 (Slepian's inequality) *If $X(t)$ and $Y(t)$ are a.s. bounded, centered Gaussian processes on T such that $Var[X(t)] = Var[Y(t)]$ for all $t \in T$ and*

$$E[X(t) - X(t')]^2 \leq E[Y(t) - Y(t')]^2 \quad \text{for all } t, t' \in T,$$

then for all $\lambda \in \mathbb{R}$

$$P\left\{\sup_{t \in T} X(t) > \lambda\right\} \leq P\left\{\sup_{t \in T} Y(t) > \lambda\right\}.$$

Proof: See eg Adler [1].

□

Theorem E.3 (Bräker) *Let $X(s)$ for $s \in S_u \subset \mathbb{R}$, where S_u is depending on the parameter u, be a locally stationary Gaussian process, ie for some $D > 0$*

$$\lim_{h\to 0}\left[\frac{E[X(s+h)-X(s)]^2}{K^2(|h|)}\right]=D,$$

where $K^2(|h|)$ *is regularly varying at 0 with index* $\alpha \in (0,2]$.

Let $G(x) := K^{-1}(1/x)$ $(x>0)$ *and* $\Delta_u(s) := G(\sqrt{D}f_u(s))$ $(s \in S_u)$, *with* $f_u(s)$ *a boundary function.*

Assume that $f_u(s)$ *satisfies the following conditions* $(f1),\ldots,(f5)$*:*

(f1) $f_u(s)$ *is continous on* S_u.

(f2) $\lim_{u\to\infty}[\inf_{s\in S_u}\{f_u(s)\}]=\infty$.

(f3) Define $\psi(x) := \frac{1}{\sqrt{2\pi x}}\exp(-\frac{x^2}{2}) \sim \Psi(x)$, *as* $x \to \infty$. $\forall\epsilon>0$ *we have*

$$\lim_{u\to\infty}\int_{S_u}\frac{\psi(\epsilon f_u(s))}{\Delta_u(s)}ds = 0.$$

(f4) Defining a sequence of functions $g_u(s,\tau) := [f_u(s+\tau\Delta_u(s))-f_u(s)]f_u(s)$ *for* $s, s+\tau\Delta_u(s) \in S_u$, *we assume that the sequence* $(g_u(s,\tau))_{u>u_0}$ *converges to some limiting function* $g(s,\tau)$, *uniformly in* $s\in S_u$, $\forall u_0 \in \mathbb{R}$ *and* τ *in compact sets of* $\mathbb{R}$, *ie*

$$\lim_{u\to\infty}\sup_{s\in S_u,|\tau|\le\theta}|g_u(s,\tau)-g(s,\tau)|=0,$$

for all θ, $\theta\in[0,\infty)$.

(f5) $\sup_{s\in S_u}|g(s,\tau)|<\infty$ *for all* $\tau\in\mathbb{R}$.

Let $\lambda_u(s) := H_\alpha 2^{-\frac{1}{\alpha}}\psi(f_u(s))\big/G(\sqrt{D}f_u(s))$ *for* $s\in S_u$, *and* l_u *(*r_u*) denote the left (right) endpoint of the interval* S_u. *(See Appendix D for the definition of* H_α*.) Define*

$$\lambda^-_{l_u} := \begin{cases}\Psi(f_u(l_u)), & \textit{if } l_u>-\infty \textit{ and } g(l_u,1)>0\\ 0 & \textit{otherwise}\end{cases}$$

$$\lambda^+_{r_u} := \begin{cases}\Psi(f_u(r_u)), & \textit{if } r_u<\infty \textit{ and } g(r_u,-1)>0\\ 0 & \textit{otherwise.}\end{cases}$$

Then

$$\lim_{u\to\infty}\frac{1}{\Lambda_u}P\left\{\exists s\in S_u : X(s)>f_u(s)\right\}=1,$$

where

$$\Lambda_u = \int_{S_u} \lambda_u(s) ds + \lambda^-_{l_u} + \lambda^+_{r_u}.$$

Proof: See Bräker [5].

□

Bibliography

[1] Adler R.J.: *An Introduction to Continuity, Extrema, and Related Topics for General Gaussian Processes*, Institute of Mathematical Statistics Hayward, California, Corollary 2.4, p.49 (1990)

[2] Bingham N.H., Goldie C.M. and Teugels J.L.: *Regular Variation*, Cambridge University Press (1987)

[3] Bingham N.H., Goldie C.M. and Teugels J.L.: *Regular Variation*, Cambridge University Press, Proposition 1.5.1, p.22 (1987)

[4] Bingham N.H., Goldie C.M. and Teugels J.L.: *Regular Variation*, Cambridge University Press, Theorem 1.5.12, p.28 (1987)

[5] Bräker H.U.: *High boundary excursions of locally stationary Gaussian processes*, PhD Thesis, Theorem 4.1, p.23 (1993)

[6] Bräker H.U.: *High boundary excursions of locally stationary Gaussian processes*, Proceedings of the Conference on Extreme Value Theory and Applications, Gaithersburg, MA, vol. 3, NIST special publ. 866, pp.69-74 (1993)

[7] Geluk J.L. and de Haan L.F.M.: *Stable Probability and their Domains of Attraction*, Discussion paper TI 97-089/4 Tinbergen Institute Rotterdam (1997)

[8] Gnedenko B.V. and Korol'uk V.S.: *Some remarks on the theory of domains of attraction of stable distributions* (in Russian), Dokl. Akad. Nauk Ukrain. SSR 4, p.275-278 (1950)

[9] Hüsler J. and Piterbarg V.: *Extremes of a certain class of Gaussian processes*, Stochastic Processes and their Applications 83, p.257-271 (1999)

[10] Piterbarg V.: *Asymptotic Methods in the Theory of Gaussian Processes and Fields*, American Mathematical Society, Theorem 8.1, p.119 (1996)

[11] Piterbarg V. and Romanova T.A.: *Numerical Estimation of the Pickand's Constant*, Universität Bern, Institut für mathematische Statistik und Versicherungslehre, technischer Bericht Nr.45 (2002)

[12] Resnick S.I.: *Extreme Values, Regular Variation and Point Processes*, Springer, Proposition 0.5, p.17 (1987)

[13] Rolski T., Schmidli H., Schmidt V. and Teugels J.: *Stochastic Processes for Insurance and Finance*, Wiley (2001)

[14] Shao Q.M.: *Bounds and estimators of a basic constant in extreme value theory of Gaussian processes*, Statistica Sinica **6**, pp.245-257 (1996)

Part II

A Heterogeneous Multi Agents Model

1 Introduction

Understanding and modeling time series is essential for the whole finance industry. Not only risk but also asset management, portfolio allocation and option pricing depend on realistic market models. For applications, we must not only understand risk but also measure and (if possible) predict it. Time series of foreign exchange (FX) rates and stock markets share various statistical properties, the so-called stylized facts. In this thesis we concentrate on major FX rates, as eg US Dollar versus Euro.

In Chapter 2 we describe how real FX markets work. We present the stylized facts of real FX data, as eg the US Dollar Swiss Franc rate of the last decade. We learn that volatility has a long memory and that the distribution of logreturns is heavy-tailed, ie the probability of extreme events is quite larger than in the Black-Scholes world, where logreturns are normally distributed. The logreturns measured over different time intervals follow certain scaling laws. Furthermore it is observed that there is an information flow from long to short horizons, ie volatility measured over large time intervals predicts volatility measured over short time intervals better than vice versa.

Since these stylized facts are so fundamental, they call for an explanation. Chapter 3 describes two popular but totally different ways to achieve this. First, time series models designed to capture some specific features are presented. They are, with the exception of the HARCH model, not very successful in reproducing all stylized facts presented in this thesis and do not model market mechanisms. So-called microstructure models do model market mechanisms in the hope to understand connections between them and time series properties.

In Chapter 4 we introduce the "Heterogeneous Multi Agents Model", which follows the heterogeneous market hypothesis [14]. This hypothesis deviates from the frequently used homogeneous market hypothesis, where all traders have the same time horizon, ie interpret news and react to news in the same way. In a heterogeneous market different groups of traders behave differently. The behavior can vary with respect to different aspects. Here we restrict ourselves to the aspect of the time horizon, ie different traders act on different time scales. There is evidence of the heterogeneous market hypothesis very indirect through certain stylized facts of financial time series. One goal of the "Heterogeneous Multi Agents Model" is to get more direct evidence for the heterogeneous market hypothesis in the sense of reproducing generic features as pointed out by LeBaron [21]. Another direction of further development of multi agents models is calibration. Here we also make a step in this direction with building the model as realistic as possible. The idea is simple: the more realistic the model is, the more stylized facts it will generate. Calibrating the model parameters in a quantitative way, this reproduction of the observed statistical properties is another goal of the "Heterogeneous Multi Agents Model". We will see that it is necessary to follow the heterogeneous market hypothesis to generate all statistical properties of real price time series.

This approach is rewarded, as the various plots in Chapter 5 show. We present a default set of model parameters that generates the stylized facts not only in a qualitative way, but also reproduces the parameters characterizing the various stylized facts. A stress-testing confirms the relative stability of the model. The “Heterogeneous Multi Agents Model” is the first microstructure model that generates both, the right scaling laws and the asymmetric information flow and thus provides a more direct support of the heterogeneous market hypothesis.

In Appendix A and B, finally, we list the full source code of the “Heterogeneous Multi Agents Model” as well as an “S-Plus” script written to analyze the resulting time series.

2 Stylized Facts of Foreign Exchange Markets

In this chapter we first have a quick look at foreign exchange (FX) markets. Then we concentrate on intraday FX rates and explain the role these high-frequency data play in finance. Afterwards, we define the variables we are working with and discuss the fact that time series of all major FX rates share some properties, so-called stylized facts. Constructing a model that reproduces these properties is a main goal of the research community as well as of this thesis.

2.1 Foreign Exchange Markets

Foreign exchange markets are the largest financial markets. The average daily turnover is about $1.2 \cdot 10^{12}$\$ [2], which is approximately the gross domestic product of France [27]. The size of the deals is usually bigger than 10^6\$. The main reason for this huge liquidity is that both exchanged assets are currencies. (FX markets involving exotic currencies are, of course, less liquid.)

Through the electronic linkages, the FX markets are virtually global. Nevertheless, there are three main components, namely East Asia, Europe and America with Tokyo, London and New York as the main trading centers.

Since there is no currency exchange, the whole FX market is over-the-counter (OTC), where deals between banks (and brokers) are negotiated over phone. There are no business hour limitations. The FX markets are open 24 hours a day, except for the weekends and a few worldwide holidays.

The FX markets may be divided into spot, forward and derivative markets. We concentrate on the spot market, where the currencies are traded at the time of transaction.

2.2 The Data and Important Variables

Market prices are tick-by-tick data, where a tick is a bid-ask quote or a transaction price, depending on the time series. FX markets are very liquid and therefore generate a huge number of ticks within a business day. Thanks to the development in computer technology, it is possible to work with these enormous data sets, that are consequently called high-frequency data. They are provided by data vendors like Reuters or Bloomberg. As opposed to low-frequency data like daily or weekly prices, high-frequency data make it possible to examine the intraday behavior of financial markets and how the markets actually work.

Having more information at one's disposal, there is the hope to have a better empirical basis to design market microstructure models.

The data we use in this thesis are kindly provided by Olsen [1] [26]. We work with the rates US Dollar (USD) versus Japan Yen (JPY) and USD versus Swiss Franc (CHF) ranging from January 1990 to June 2001 and USD against Euro (EUR) as well as EUR against CHF ranging from January 1999 to June 2001. The data are given in the form of *logarithmic middle prices* $\tilde{p}_t$

$$\tilde{p}_t := \frac{1}{2}\left[\log(p_{\mathrm{ask},t}) + \log(p_{\mathrm{bid},t})\right] = \log\sqrt{p_{\mathrm{ask},t}\, p_{\mathrm{bid},t}}, \tag{1}$$

where $t = 0, 1, 2, \ldots$ are discrete times of new quotes measured in 1 hour θ-time. θ-time [8] is an activity related time scale, which removes daily and weekly seasonalities. This procedure is necessary for the analysis of more subtle effects. Seasonalities are due to fluctuations in the market activity, the number of deals within a certain time interval. The idea of θ-time is to lengthen the physical time when market activity is high and to shorten the physical time when market activity is low. In this thesis we always use θ-time.

FX prices can be looked at from two sides: the value of an US Dollar can be measured in Swiss Franc (USD/CHF) and vice versa. Equation (1) shows that if $\tilde{p}_t$ is the logarithmic middle price of 1 USD expressed in CHF, the logarithmic middle price of 1 CHF expressed in USD is given by $-\tilde{p}_t$.

A FX rate, ie the *price* p_t, is given by $p_t := \exp(\tilde{p}_t)$. In the case of transaction prices, $\tilde{p}_t$ has to be replaced by $\log(p_{transaction,t})$.

We mainly work with the time series of *logreturns* $(r_{t,\Delta t})$, defined by

$$r_{t,\Delta t} := \log\left(\frac{p_t}{p_{t-\Delta t}}\right),$$

with Δt a time interval of fixed size measured in θ-time. To simplify the notation we often write r_t instead of $r_{t,\Delta t}$. The logreturns have the advantage that they are free of units and are therefore comparable among different time series. Further, it is believed that the time series (r_t), which, using a Taylor series argument, for small relative price changes is close to the time series of relative returns $[(p_t - p_{t-\Delta t})/p_{t-\Delta t}]$, can be modeled by a stationary stochastic process.

The *volatility* $v_{t,\Delta t}$, is defined as the absolute logreturns

$$v_{t,\Delta t} := |r_{t,\Delta t}|.$$

[1] www.olsen.ch

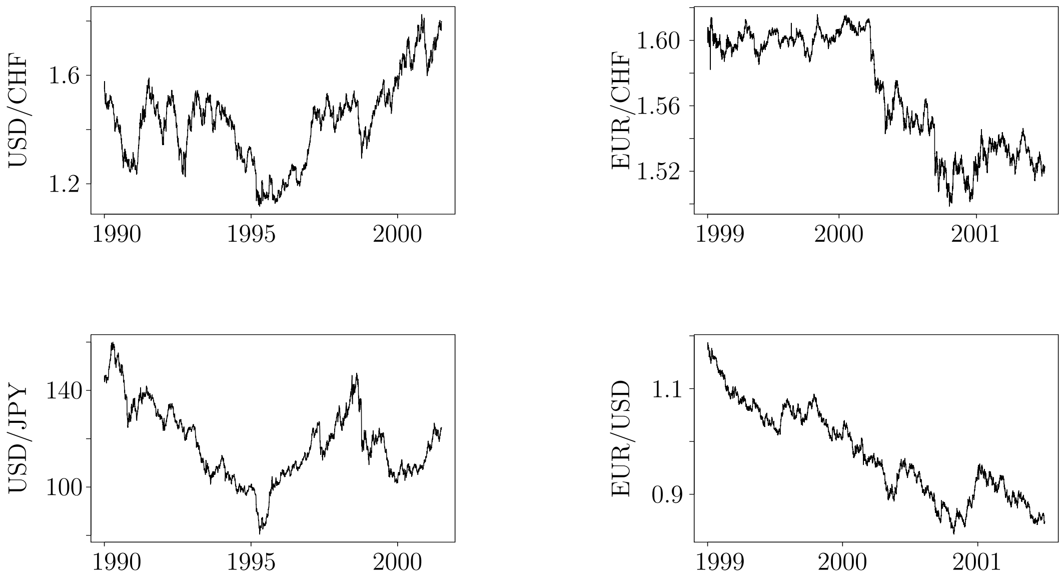

Figure 2.1: Various FX rates. The year marks indicate the beginning of a calendar year.

The choice of the time interval Δt to calculate the logreturns and the volatility is a crucial point. Traders with different time horizons Δt observe different volatilities. This will be discussed in detail in Chapter 2.5.

It is well-known that the time series (r_t) of various FX rates share some statistical properties. We demonstrate these so-called *stylized facts* using the rates USD/CHF, USD/JPY, EUR/CHF and EUR/USD in the following chapters. The corresponding price time series are plotted in Fig. 2.1.

2.3 Long Memory and Volatility Clustering

We are interested in the time series of logreturns (r_t) plotted in Fig. 2.2. The time interval Δt over which the logreturns are measured is one hour in θ-time.

Large and small values of r_t occur in clusters, which means that the variance of the logreturns is not constant. This *volatility clustering* (see Fig. 2.2) shows that periods of quiet markets are interrupted by periods of large volatility.

To make more quantitative statements we examine the autocorrelation function (acf) of the logreturns and of the volatility. The acf of an arbitrary time series x_i, $\rho(\tau)$, is defined as

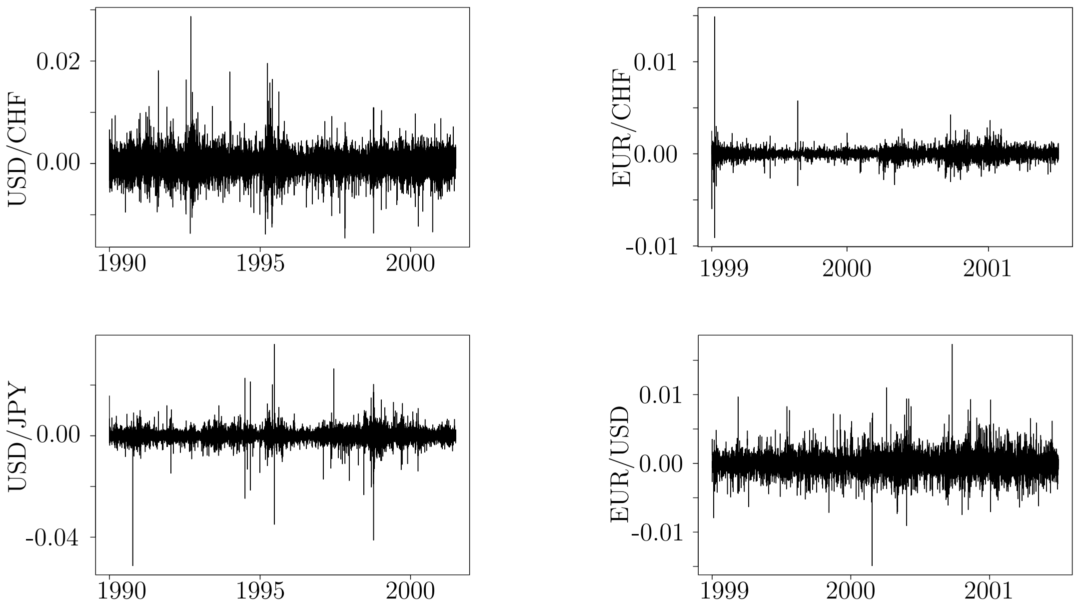

Figure 2.2: The logreturns of all FX rates are clustered, ie, periods of quiet markets are interrupted by periods of large price movements.

$$\rho(\tau) := \frac{\sum_{i=1}^{n-\tau}(x_{i+\tau} - \bar{x})(x_i - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

where τ is the lag and n the length of the time series, furthermore $\tau < n$, $\bar{x} := \sum_{i=1}^{n} x_i$.

Let us have a look at the acf of the logreturns r_t and of the volatility v_t. We note that the sample autocorrelations of the logreturns r_t are negligible at almost all lags, as can be seen in Fig. 2.3. The price process p_t seems to follow a random walk.

But the sample autocorrelations of the volatility v_t are different from zero for a large number of lags (see Fig. 2.4). The fact that financial time series exhibit a long memory is due to the volatility clustering.

To make possible a more quantitative examination of the acf, we introduce the concept of short and long memory, which does only make sense when considering a stationary time series. It is believed that the time series of absolute logreturns ($|r_t|$) is stationary. Since the acf value corresponding to lag 0 is 1, we omit this value in our analysis.

A time series is said to have a *short memory* if the acf decays at an exponential rate

$$\rho_e(\tau) = ke^{-\tau/c},$$

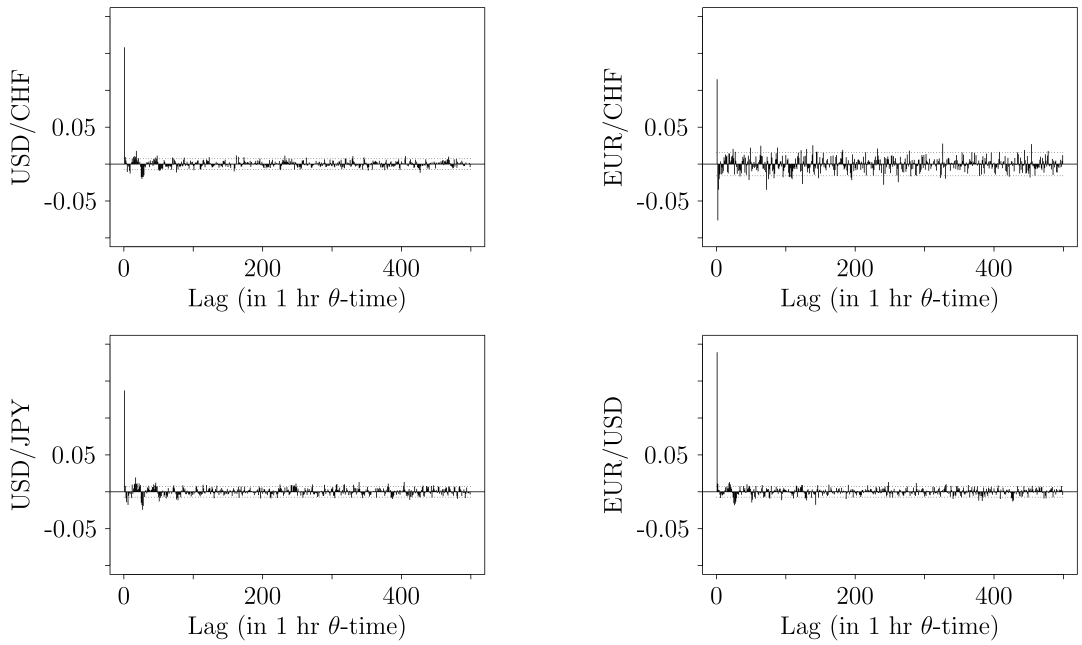

Figure 2.3: The autocorrelation function of the logreturns is mostly within the 95% line of white noise. The value corresponding to lag 0 is not plotted.

where τ is the lag and $k, c > 0$ are constants.

We say that a time series has a *long memory* if

$$\sum_{\tau=0}^{\infty} |\rho(\tau)| = \infty, \tag{2}$$

where $\rho(\tau)$ is the acf depending on the lag τ.

This is true, eg, if the acf decays at a hyperbolical rate,

$$\rho_h(\tau) = c\tau^{-h} \tag{3}$$

for some constants $c, h > 0$. The bigger h, the faster $\rho(\tau)$ tends to 0 as $\tau \to \infty$.

To find out whether the time series of the volatility v_t, the absolute logreturns $|r_t|$, has a short or a long memory we use the following two step procedure:

First step: Plotting the acf $\rho(\tau)$, we get a first impression. Then we fit $\rho(\tau)$ to

$$\rho_g(\tau) = m(1 + n\tau)^{-1/n}$$

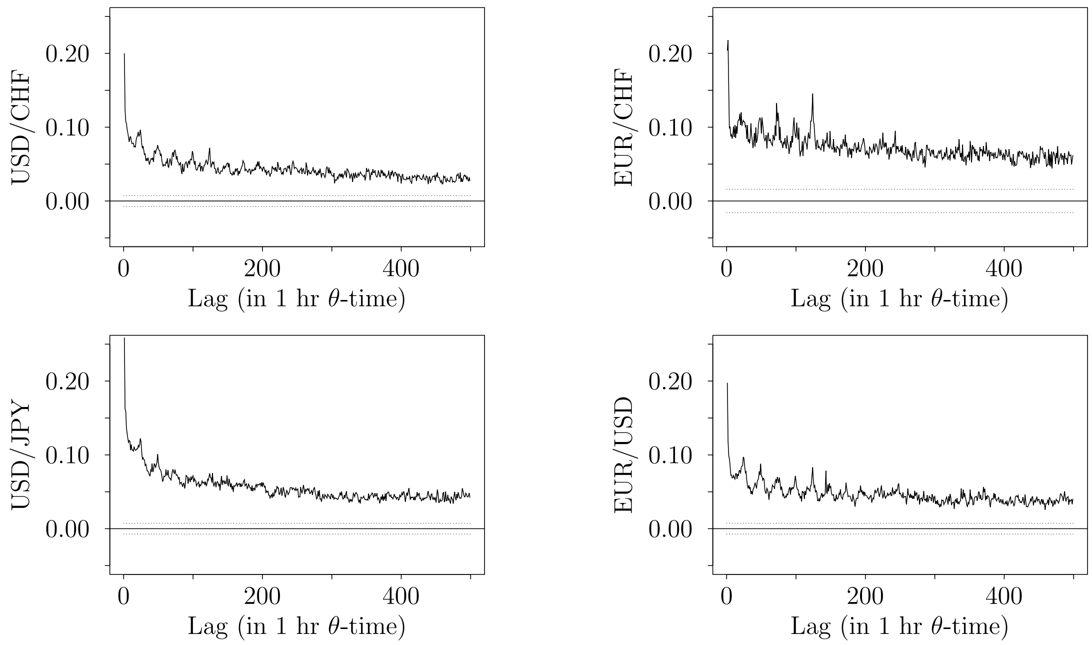

Figure 2.4: The autocorrelation function of the volatility (absolute logreturns) decays at a hyperbolic rate for all FX rates. Ie, FX markets have a long memory. The dotted lines indicate the 95% interval for white noise. The value corresponding to lag 0 is not plotted.

For $n \approx 0$ the model reduces to $\exp(-\tau/a)$ and the time series has a short memory.

Second step: If the hypothesis that the time series has a short memory can be rejected, ie, if 0 lies outside the 95% confidence interval of n, we fit the acf to the hyperbolic model (3). We use usual linear regression to estimate c and h in

$$\log(\rho_h(\tau)) = \log(c) - h \log(\tau).$$

The estimated values of n and h for our FX rates are given in Tab. 2.1. Since n is different from zero for all rates, we conclude that the acf decays at a hyperbolic rate and not at an exponential rate. FX rates do have a long memory.

2.4 Heavy Tails

A very important aspect for risk-management is the fact that the distributions of logreturns are increasingly fat-tailed for decreasing time intervals Δt. That the distribution is not normal can be seen in Fig. 2.5, where the quantiles of the logreturns plotted against the Normal distribution have the shape of an “s”. This means that extreme events occur with

Rate	n	h
EUR/CHF	5.9224 ± 0.1843	0.1678 ± 0.0051
EUR/USD	2.5376 ± 0.1005	0.3887 ± 0.0140
USD/CHF	3.7457 ± 0.0582	0.2627 ± 0.0039
USD/JPY	3.5943 ± 0.0395	0.2730 ± 0.0029

Table 2.1: Fitting the autocorrelation function of the absolute logreturns to $\rho_g(\tau) = m(1+n\tau)^{-1/n}$ and $\rho_h(\tau) = c\tau^{-h}$.

a higher probability than in the frequently used Gaussian models, especially in the option pricing model of Black and Scholes [6] and Merton [24]. Volatility clustering, respectively the long memory effect, is a reason for these *heavy tails*, since large price changes rarely occur alone.

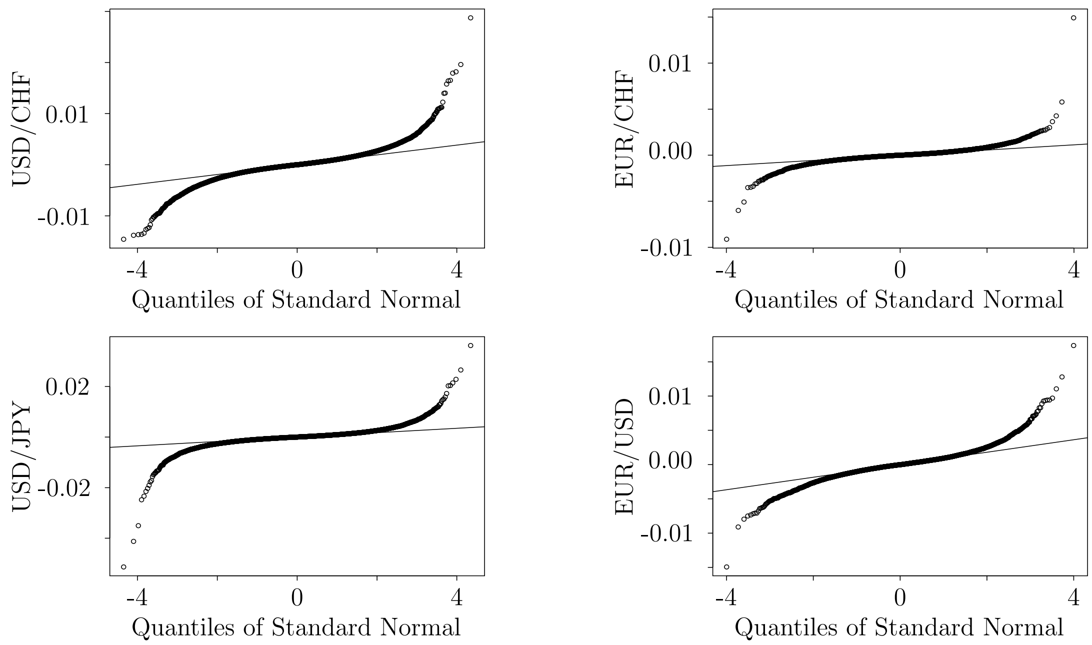

Figure 2.5: QQ-plots of the logreturn data against the standard Normal distribution reveal heavy tails for all FX rates.

It does not suffice to know that the data is heavy-tailed. We need to know the tail index, which describes the shape of the tail of the distribution and thus the probability of extreme events. We define and estimate the tail index using a peaks-over-threshold (POT) model, which describes the distribution of the values exceeding a high threshold. It is based on the generalized Pareto distribution (GPD), which is determined by the scaling parameter β and the shape parameter ξ. $\alpha := 1/\xi$ is called *tail index*. The GPD is given by

$$G_{\xi,\beta}(x) = \begin{cases} 1-(1+\xi x/\beta)^{-1/\xi} & (\xi \neq 0), \\ 1-\exp(-x/\beta) & (\xi = 0), \end{cases}$$

where $\beta > 0$ and $x \geq 0$ for $\xi \geq 0$ and $0 \leq x \leq -\beta/\xi$ for $\xi < 0$. We may add a location parameter μ. The GPD $G_{\xi,\mu,\beta}(x)$ is defined to be $G_{\xi,\beta}(x-\mu)$.

If $\xi > 0$, $G_{\xi,\beta}(x)$ is heavy-tailed and we just have a reparametrized version of the ordinary Pareto distribution, where a larger ξ (smaller α) indicates fatter tails. The case $\xi = 0$ corresponds to an exponential distribution and $\xi < 0$ is known as Pareto type II distribution.

We now have a look at the basic ideas of the POT model.

Let $F(x) := P\{X \leq x\}$ be the distribution function of a random variable X. The distribution of excesses over a high threshold u is defined by

$$F_u(y) := P\{X - u \leq y | X > u\} = P\{X \leq y + u | X > u\}$$

for $0 \leq y < x_F - u$, where $x_F := \sup\{x | F(x) < 1\}$ is the right endpoint of F. That is, $F(x) < 1$ $(x < x_F)$ and $F(x) = 1$ $(x \geq x_F)$. $F_u(y)$ is the probability that a loss exceeds the threshold u by maximal y, given that it exceeds the threshold u.

For $y > u$, $F_u(y)$ may be expressed in terms of $F(u)$,

$$F_u(y) = \frac{F(y+u) - F(u)}{1 - F(u)}. \tag{4}$$

In the following we will need some results of extreme value theory. Let $X_1, X_2, \ldots$ be independent, identically distributed random variables with unknown underlying distribution function $F(x) = P\{X_i \leq x\}$. The generalized extreme value (GEV) distribution function is given by

$$H_{\xi,\beta}(x) = \begin{cases} \exp(-(1+\xi x/\beta)^{-1/\xi}) & (\xi \neq 0), \\ \exp(-e^{-x/\beta}) & (\xi = 0), \end{cases}$$

where x is such that $1 + \xi x > 0$. ξ and β are the shape parameter and scaling parameter, respectively.

We say that a distribution function $F(x)$ is in the domain of attraction of $H_{\xi,\beta}(x)$ ($F \in$ MDA$(H_{\xi,\beta}(x))$), if for $M_n := \max(X_1, \ldots, X_n)$ there exists a sequence of real numbers $a_n > 0$ and b_n such that

$$P\{(M_n - b_n)/a_n \le x\} = F^n(a_n x + b_n) \xrightarrow{d} H(x),$$

as $n \to \infty$, where $\xrightarrow{d}$ means convergence in distribution.

We emphasize that all common continuous distributions of actuarial science like Gauss, lognormal, χ^2, t, F, gamma, exponential, uniform or beta are in the domain of attraction of the GEV.

We give a main result of extreme value theory.

Theorem 2.1 *There exists a positive function $\beta(u)$ such that*

$$\lim_{u \to x_F} \sup_{0 \le y < x_F - u} |F_u(y) - G_{\xi,\beta(u)}(y)| = 0$$

if and only if $F \in MDA(H_{\xi,\beta}(x))$.

The GPD is thus the natural model to approximate the unknown excess distribution $F_u(y)$ above a sufficiently high threshold u.

Consequently, in the POT model we set

$$F_u(y) = G_{\xi,\beta}(y).$$

With (4) and defining $x := y + u$, the underlying distribution $F(x)$ is then

$$F(x) = (1 - F(u))G_{\xi,\beta}(x - u) + F(u),$$

for $x > u$. We denote the sample size by n and the number of excesses over the threshold u by N_u. Obviously, $(n - N_u)/n$ serves as an estimator for $F(u)$.

Using maximum likelihood estimators for ξ and β we get the tail estimator

$$\begin{aligned}
\hat{F}(x) &= \left(1 - \frac{n - N_u}{n}\right) G_{\xi,\beta}(x - u) + \frac{n - N_u}{n} \\
&= \frac{N_u}{n}\left(1 - \left(1 + \frac{\xi(x-u)}{\beta}\right)^{-\frac{1}{\xi}}\right) + \frac{n - N_u}{n} \\
&= 1 - \frac{N_u}{n}\left(1 + \frac{\hat{\xi}(x-u)}{\hat{\beta}}\right)^{-\frac{1}{\hat{\xi}}},
\end{aligned}$$

where $x > u$.

There is an "S-Plus" module written by McNeil [23] to estimate the shape parameter ξ and hence the tail index $\alpha = 1/\xi$.

In Tab. 2.2 we list the tail indices of our four time series estimated with the POT-model and using the largest 5% of the logreturns.

Rate	α
EUR/CHF	2.86 ± 0.47
EUR/USD	4.62 ± 0.90
USD/CHF	4.49 ± 0.42
USD/JPY	3.65 ± 0.28

Table 2.2: Tail indices estimated using the POT model. The lower value for the EUR/CHF rate indicates a larger probability of extreme events than for the other rates.

2.5 Scaling Laws

Since there is no privileged time interval Δt at which one should examine the logreturns, it is important to study the dependence of the volatility on the frequency of its measurement.

The *scaling laws* describe the relation between the average volatility, measured as a power p of the absolute logreturns $|r_t| = |\log(p_t/p_{t-\Delta t})|$, and the time interval Δt over which they are measured. They are given by

$$\left\{E\left[|r_{t,\Delta t}|^p\right]\right\}^{1/p} = c_p(\Delta t)^{D_p}, \quad (p = 1, 2, 3, \ldots) \tag{5}$$

where $E[\cdot]$ denotes the empirical mean and c_p and D_p, the scaling or drift exponent, are parameters depending on p. Taking the logarithm of (5) we get

$$\log\left(\left\{E\left[|r_{t,\Delta t}|^p\right]\right\}^{1/p}\right) = \log(c_p) + D_p \log(\Delta t).$$

The scaling laws are indicated by a straight line when plotted on a double logarithmic scale and D_p is estimated using the method of least squares. In the case of Gaussian processes we have $D_p = 1/2$. This linear regression is only an approximation since the $E[|r_{t,\Delta t}|^p]$ are not totally independent for different Δt.

In Figs. 2.6, 2.7 and 2.8, we see the scaling laws of our FX rates for $p = 1$, $p = 2$ and $p = 3$. The corresponding scaling exponents are listed in Tab. 2.3. The fact that D_1, D_2 and D_3 for EUR/CHF are smaller (and not larger) than those of the other rates indicates that in the case of EUR/CHF the size of the absolute logreturns measured over large time intervals are successfully reduced with respect to the short-term volatility. Obviously, CHF is pegged to EUR. See Dacorogna [15], who considered FX rates of the no longer existing European Monetary System, introduced in the 1990s to keep some intra-European rates within some bands, for a similar discussion.

Rate	D_1	D_2	D_3
EUR/CHF	0.4154 ± 0.0016	0.4285 ± 0.0007	0.4205 ± 0.0005
EUR/USD	0.5329 ± 0.0005	0.5040 ± 0.0002	0.4829 ± 0.0003
USD/CHF	0.5389 ± 0.0003	0.5171 ± 0.0004	0.4976 ± 0.0005
USD/JPY	0.5516 ± 0.0004	0.5310 ± 0.0005	0.4977 ± 0.0006

Table 2.3: Estimation of the scaling exponents D_1, D_2 and D_3 for real FX data. We note the deviating values for the EUR/CHF rate, which might be an indicator for interventions of, probably, the Swiss central bank.

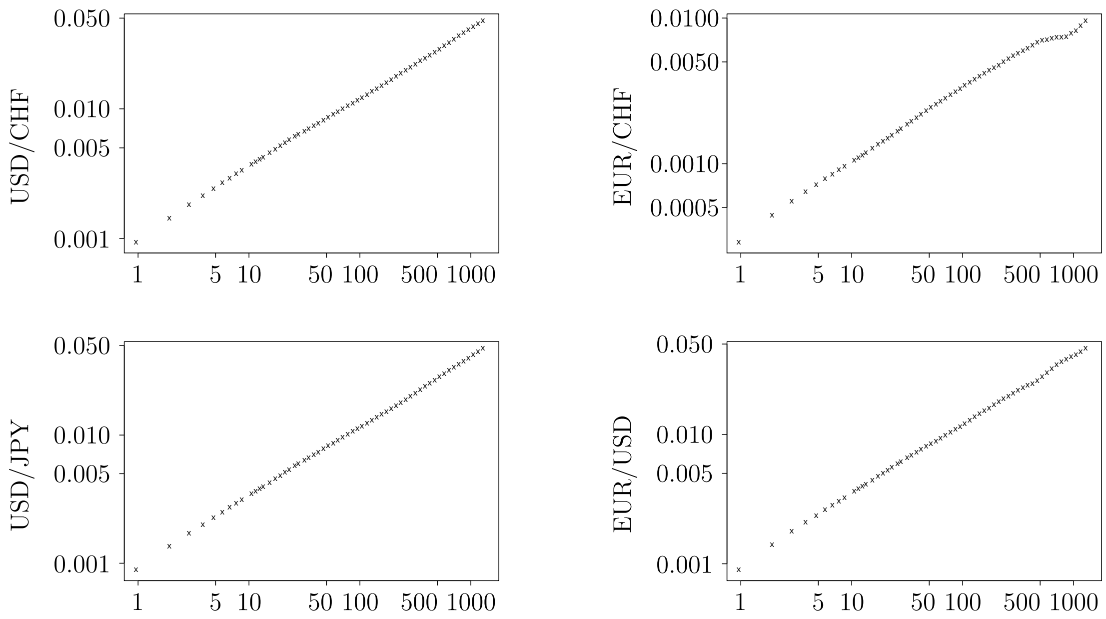

Figure 2.6: The scaling law for $p = 1$. The mean absolute logreturns as a function of the time horizon (in 1h θ -time) over which they are measured on a log-log scale. Not all points are plotted.

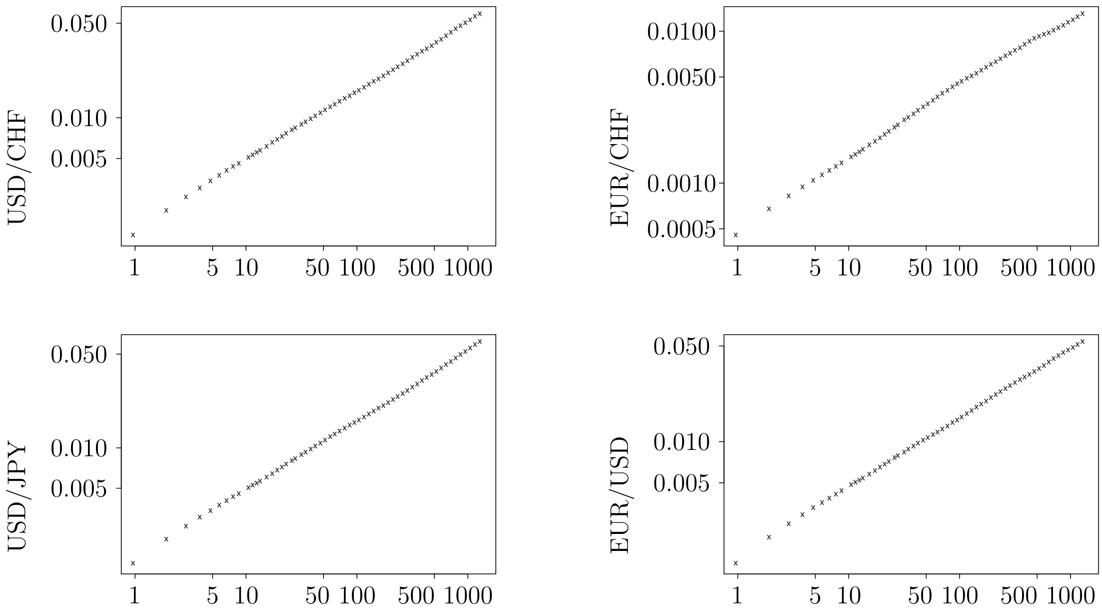

Figure 2.7: The scaling law for $p = 2$. Not all points are plotted.

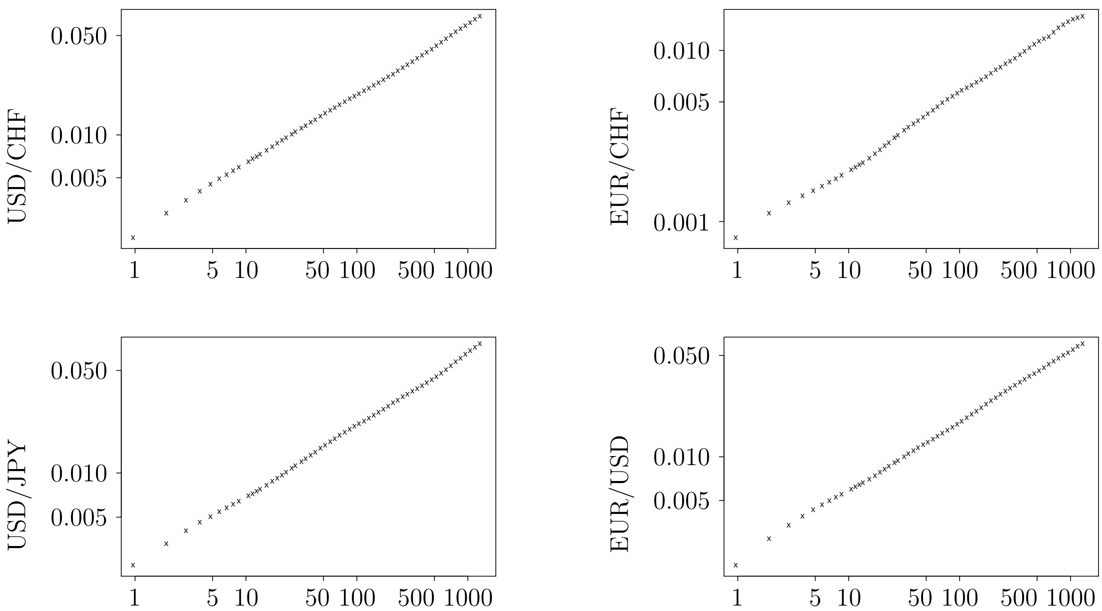

Figure 2.8: The scaling law for $p = 3$. Not all points are plotted.

2.6 Asymmetric Information Flow

We already mentioned that there is no privileged time interval Δt when measuring the volatility. So it is natural to compare volatilities measured on a fine time grid with those measured on a coarse grid. Lagged correlation ρ_τ, which compares two time series not only simultaneously but also with a time shift τ, reveals causal relations. If the deviations between ρ_τ and $\rho_{-\tau}$ become significant, there is an asymmetry in the information flow.

The formula for the lagged correlation between the time series of fine f_i $(i = 1, \ldots, n)$ and course volatility c_i $(i = 1, \ldots, n)$ is

$$\rho_\tau(f, c) = \frac{\sum_{i=1}^{n-\tau}(f_i - \bar{f})(c_{i+\tau} - \bar{c})}{\sqrt{[\sum_{i=1}^{n-\tau}(f_i - \bar{f})^2][\sum_{i=1}^{n-\tau}(c_{i+\tau} - \bar{c})^2]}},$$

where $\bar{f} = 1/(n-\tau)\sum_{i=1}^{n-\tau} f_i$ and $\bar{c} = 1/(n-\tau)\sum_{i=1}^{n-\tau} c_{i+\tau}$ for $\tau \geq 0$. The lag τ is an integer. We use the relation $\rho_{-\tau}(f, c) = \rho_\tau(c, f)$.

In the case of our FX rates, we choose the fine volatility to be the absolute hourly logreturns averaged over 8 observations, so covering an 8-hour interval in θ-time. Coarse volatility is then the absolute logreturn over a full 8-hour interval. We then delay the time series of fine volatility versus the time series of coarse volatility and vice versa and calculate the difference between the two lagged correlation functions. The results are shown in Tab. 2.4 and Fig. 2.9, where the 95% confidence interval of white noise are determined using a Monte Carlo method in the following way: Since we have at about 15000 tick prices for the time series involving the Euro and at about 70000 tick prices for the other time series, we first draw 15000, respectively 70000, random numbers from a standardized Gauss distribution, representing a white noise time series. We calculate then the lagged correlations ρ_1 and ρ_{-1} as well as $\rho_1 - \rho_{-1}$, as described above. We repeat these steps 50000 times and cut the upper and lower 2.5% to get simulated confidence intervals. The intervals are $[-0.0614, +0.0609]$ for the shorter and $[-0.0284, +0.0281]$ for the longer time series. For later use we give the confidence interval corresponding to a time series of length 100000. It is given by $[-0.0237, +0.0235]$.

In Fig. 2.9 we see that the time series involving the Euro are for the first few lags at the border of the 95% confidence interval of white noise, whereas the other two FX rates are significantly outside of the confidence intervals. The values for the difference of the lagged correlations, however, are more or less the same for all four time series (see Tab. 2.4). We therefore conclude that the values for the lagged correlation functions hardly change when we increase the length of the time series, whereas the simulated confidence intervals do decrease. We see in Fig. 2.9 that the correlation function where coarse volatility is lagged (back in time) compared to fine volatility (negative lags) is larger than when fine volatility is lagged (back in time) compared to coarse volatility (positive lags). Coarse volatility, which captures the view of long-term traders, predicts the fine volatility, which

time lag	EUR/CHF	EUR/USD	USD/CHF	USD/JPY
1	-0.0567	-0.0027	-0.0410	-0.0674
2	-0.0320	-0.0852	-0.0837	-0.0785
3	-0.0700	-0.0777	-0.0901	-0.1121
4	-0.0306	-0.0125	-0.0311	-0.0204
5	-0.0239	-0.0397	-0.0236	-0.0358
6	-0.0237	-0.0949	-0.0452	-0.0438
7	-0.0206	-0.0064	-0.0110	-0.0178
8	-0.0105	-0.0175	-0.0259	-0.0221
9	-0.0182	-0.0509	-0.0574	-0.0423
10	-0.0196	0.0169	-0.0158	-0.0120
11	-0.0093	-0.0538	-0.0096	-0.0224
12	0.0098	-0.0477	-0.0368	-0.0320
13	0.0122	-0.0178	-0.0183	0.0022
14	-0.0407	-0.0478	-0.0009	0.0087
15	0.0034	-0.0101	-0.0305	-0.0038

Table 2.4: Difference between lagged correlation. Negative values indicate the predictability of finely defined volatility from coarse volatility. The time lags are multiples of 8 hr in θ time. The 95% confidence intervals are given by $[-0.0614, +0.0609]$ for EUR/CHF and EUR/USD and $[-0.0284, +0.0281]$ for USD/CHF and USD/JPY.

captures the view of short-term traders, better than the other way round. We observe an *asymmetric information flow* from long to short time horizons.

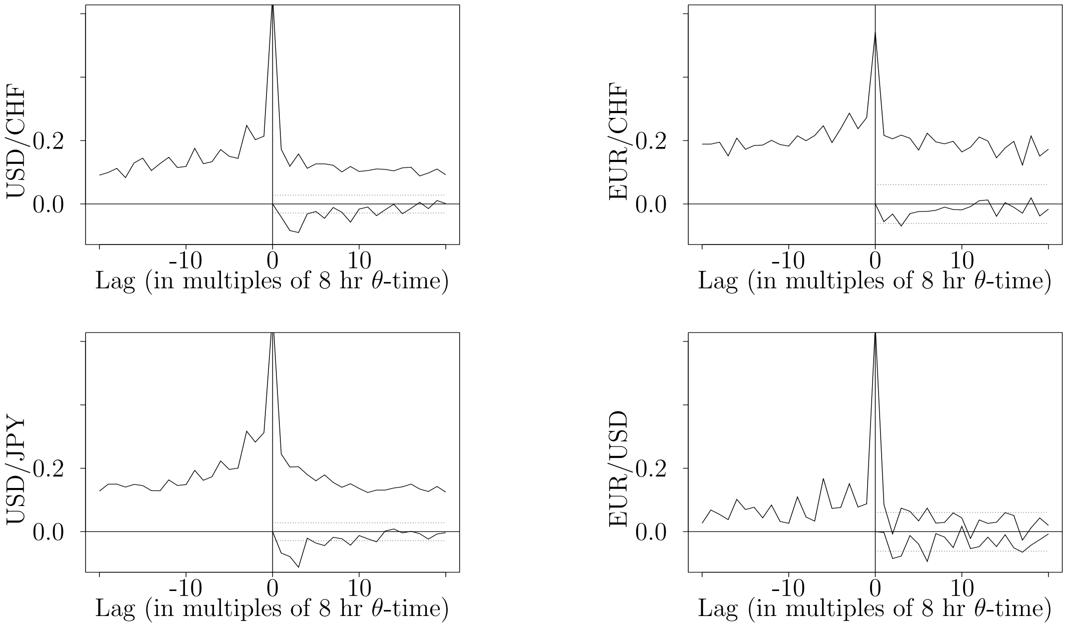

Figure 2.9: Asymmetric lagged correlation function of fine and coarse volatilities. In the left branch of the upper curve, the correlation of coarse and further in the future lying fine volatility is plotted, ie $\rho_{-\tau}$ and ρ_τ in the upper right branch. The lower curve, representing $\rho_\tau - \rho_{-\tau}$, indicates the asymmetry. Coarse volatility predicts fine volatility better than vice versa. The dotted lines indicate the simulated 95% confidence interval of white noise.

3 Modeling FX Markets

To explain the stylized facts of FX markets mentioned in Chapter 2 there are time series and multi agents models. We first present an overview of the most important time series models (econometric models), which are built on past prices and volatilities, before we discuss multi agents models, which emphasize certain aspects of the traders. These two approaches are not of the same value. Traditionally, time series models are used. Their techniques are much older than those of multi agents models, that depend on computer technology. However, time series models do not give insights into the market mechanisms, a gap filled by the multi agents models that become more and more popular.

3.1 Time Series Modeling

Researchers first examined autoregressive (AR), autoregressive moving average (ARMA) and its derivations like autoregressive integrated moving average (ARIMA) models, which all have a constant variance. With the availability of high frequency data, it turned out that the volatility is not constant. So the early models were discarded and, in this thesis, we concentrate on more realistic models. In practice, most models are now based on the following formula for the logreturns r_t,

$$r_t = \sigma_t \epsilon_t \quad (t = 0, 1, \ldots), \tag{6}$$

where the time dependent volatility $\sigma_t > 0$ is independent of the i.i.d. random variable ϵ_t, which has zero mean and variance 1. The key question is how to model the volatility variable σ_t.

One class are *autoregressive conditional heteroscedastic models* (ARCH) that define the volatility σ_t of the logreturns r_t as a function of past logreturn values $r_{t-1}, r_{t-2}, \ldots$. This function may be simple, as in the original model of Engle [17]

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i r_{t-i}^2,$$

where $\alpha_0 > 0$, $\alpha_i \geq 0$ and $\alpha_p > 0$. The corresponding model is called autoregressive conditionally heteroscedastic of order p, or ARCH(p).

Bollerslev [7] introduced *generalized ARCH* processes of order (p,q), or GARCH(p,q), where

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i r_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2,$$

where $\alpha_0, \beta_0 > 0$, $\alpha_i, \beta_j \geq 0$ and $\alpha_p, \beta_q > 0$. Here, σ_t depends also on its own past values. However, there exists always an equivalent formulation that defines σ_t only by past logreturns, but it requires infinitely many coefficients α_i. GARCH(p,q) can reproduce the heavy tails observed in daily financial time series but it is not able to explain the other stylized facts of real data.

The *heterogeneous autoregressive conditional heteroscedasticity* model, or HARCH(n) model, of Müller et al [25] is based on the hypothesis of a heterogeneous market. Ie, the volatility process σ_t is modeled using logreturns $r_{t,\Delta t}$ measured over time intervals Δt of different sizes. They suggest as volatility process

$$\sigma_t^2 = c_0 + \sum_{j=1}^{n} c_j r_{t,\Delta t=j}^2,$$

where $c_0 > 0, c_n > 0$ and $c_j \geq 0$ for $j = 1, \ldots, n-1$. The coefficients $c_1, c_2, \ldots$ cannot be chosen totally free, since the heterogeneous market approach leads to some constraints. HARCH(n) processes have a long memory and do generate an asymmetric information flow.

Lévy processes, introduced by Barndorff-Nielsen [3] in an investigation of the physics of wind-blown sand, take into account the heavy tails in the distribution of the logreturns. The density of the *generalized hyperbolic distribution* is given by

$$\begin{aligned} f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu) :=& \alpha(\lambda, \alpha, \beta, \delta)(\delta^2 + (x-\mu)^2)^{(\lambda - \frac{1}{2})/2} \times \\ & \times K_{\lambda - \frac{1}{2}} \left(\alpha \sqrt{\delta^2 + (x-\mu)^2} \right) \exp\left(\beta(x-\mu)\right), \end{aligned} \tag{7}$$

where

$$\alpha(\lambda, \alpha, \beta, \delta) := \frac{(\alpha^2 - \beta^2)^{\lambda/2}}{\sqrt{2\pi}\alpha^{\lambda - \frac{1}{2}} \delta^\lambda K_\lambda \left(\delta \sqrt{\alpha^2 - \beta^2} \right)}$$

is the normalizing constant and K_ν the modified Bessel function of the third kind with index ν. K_ν can be represented as

$$K_\nu(z) = \frac{1}{2} \int_0^\infty y^{\nu - 1} \exp\left(-\frac{1}{2} z(y + y^{-1}) \right) dy.$$

The density depends on 5 parameters: $\alpha > 0$ determines the shape, β with $0 \leq |\beta| < \alpha$ the skewness, $\mu \in \mathbb{R}$ the location , $\delta > 0$ the scale and $\lambda \in \mathbb{R}$ characterizes certain subclasses, ie the heaviness of the tails. With these 5 parameters it is possible to fit the empirical densities in an optimal way, different to the case of the normal density, which has only 2 parameters. Eberlein [16] introduced the subclass of hyperbolic distributions, defined by $\lambda := 1$, and Barndorff-Nielsen [4] the normal inverse Gaussian distribution, characterized by $\lambda := -\frac{1}{2}$, in the context of finance.

Barndorff-Nielsen and Halgreen [5] showed that generalized hyperbolic distributions are infinitely divisible. Therefore every member of this distribution family with parameters $(\lambda, \alpha, \beta, \delta, \mu)$ generates a Lévy process X_t $(t \geq 0)$, ie a process with stationary independent increments such that $X_0 = 0$ and the distribution of X_1 has the density (7). This is the same procedure as in the case of Brownian motion, a Gaussian process, which is generated by the normal distribution. Lévy processes can be represented in the form

$$X_t = \sigma B_t + Z_t + \nu t,$$

where $\sigma^2 \geq 0$, $\nu \in \mathbb{R}$, B_t a standard Brownian motion and Z_t a purely discontinuous martingale independent of B_t. We note that Brownian motion is itself a Lévy process. But whereas a Brownian motion has continuous sample paths, generalized hypberbolic Lévy processes change their values by jumps only.

A second class of FX models are *stochastic volatility models*, where σ_t has its own stochastic component. As opposed to ARCH and GARCH models, σ_t does not depend on past logreturns but on its own past values. A problem is that the volatility σ_t is not directly observable. Thus, it is more difficult to estimate the parameters of stochastic volatility models than those of ARCH and GARCH models.

We consider the Cascade Model of Breymann et al [9], which is based on an analogy between FX markets and hydrodynamical turbulence in physics, cf Ghashghaie et al [19].

We make a brief excursion into the theory of fully developed turbulence according to Kolmogorov. It is observed that turbulence generates eddies at many different length scales. Eddies now break up into smaller eddies, which themselves break up into even smaller ones and so on. Energy is transferred from big to small eddies until at the smallest length scales the energy is finally dissipated. One possibility to describe this energy cascade, known as Richardson cascade, is to use the Random Cascade Model explained in detail in Frisch [18]. He starts with a cube of side l_0 in which the energy dissipation is taken to be uniform. $\epsilon > 0$ is its value per unit volume. That way generation $n = 0$ is defined. To obtain generation $n = 1$, the cube is subdivided into 8 equal cubes of side $l = l_0/2$. In each subcube the dissipation is multiplied by independent realizations of a random variable W, which suffices

$$W \geq 0, \quad E[W] = 1, \quad E[W^q] < \infty \ (q > 0).$$

At the nth generation there are 2^{3n} cubes of side $l = l_0 2^{-n}$ and the dissipation per unit volume is given by

$$\epsilon_l = \epsilon \prod_{k=1}^{n} W_k,$$

where the W_ks are independently and identically distributed.

The analogy between turbulence and FX markets in the Cascade Model is in the following way: Energy and spatial distance correspond to information and time. An energy cascade in space hierarchy then corresponds to an information cascade in time hierarchy.

We go back to the Cascade Model. The underlying logreturn equation is again (6), $r_t = \sigma_t \epsilon_t$, where ϵ_t are i.i.d. Student random numbers with three degrees of freedom. The main building block is the multiplicative stochastic cascade for the volatility σ_t measured on various time horizons $\tau^{(k)}$, where k is the level of the hierarchy, representing the heterogeneity of the market. We assume that $\tau^{(k)}/\tau^{(k-1)} =: p < 1$ is independent of k. At each level k of the hierarchy there is a volatility $\sigma_t^{(k)}$. They are recursively defined by

$$\sigma_t^{(k)} := a_t^{(k)} \sigma_t^{(k-1)}, \tag{8}$$

where $a_t^{(k)}$ are time dependent random factors. Assuming that the volatility $\sigma^{(0)}$ on the largest horizon is time independent, the volatility on the shortest time horizon m determines the logreturns in (6). With $\sigma_t := \sigma_t^{(m)}$ and (8) we get

$$\sigma_t = \sigma^{(0)} \prod_{k=1}^{m} a_t^{(k)}.$$

The random factors $a_t^{(k)}$ suffice the following renewal scheme: At time $t = 0$, $a_0^{(k)}$ ($k = 1, \ldots, m$) are drawn from independent lognormal distributions $LN(a_k, \lambda_k^2)$. For $t > 0$, $a_t^{(1)}$ maintains the value $a_{t-1}^{(1)}$ with probability $1 - w^{(1)}$ and else is drawn from $LN(a_1, \lambda_1^2)$. In the latter case all subsequent factors $a_t^{(k)}$ ($k > 1$) are also renewed, ie drawn from $LN(a_k, \lambda_k^2)$. If $a_t^{(1)} = a_{t-1}^{(1)}$, $a_t^{(2)}$ will be renewed with probability $w^{(2)}$. These rules are applied through the whole cascade down to $k = m$. The renewal probabilities $w^{(k)}$ are given by

$$w^{(k)} := 1 - \frac{1 - p^{m-k}}{1 - p^{m-k+1}} \quad (k = 1, 2, \ldots, m).$$

A typical run with 100000 prices is shown in Figs. 3.1 and 3.2. The Cascade Model successfully captures the volatility clustering, long memory ($h = 0.2644 \pm 0.0027$), heavy tails ($\alpha = 3.41 \pm 0.20$) and scaling laws ($D_1 = 0.5503 \pm 0.0005, D_2 = 0.5320 \pm 0.0006, D_3 = 0.5039 \pm 0.0008$). It does not, however, generate an asymmetric information flow.

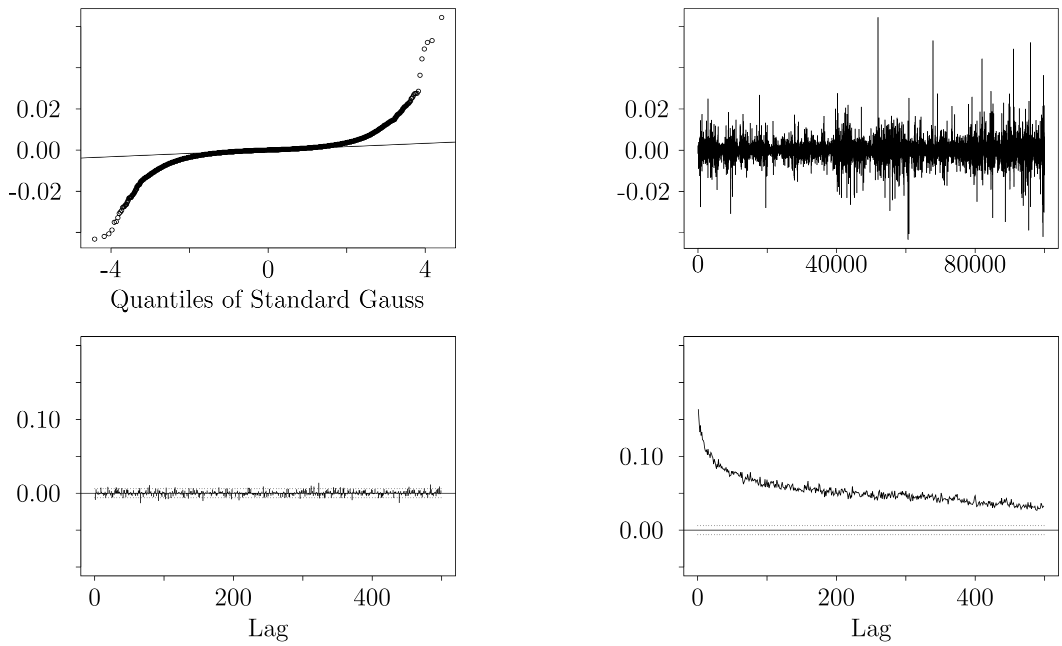

Figure 3.1: The Cascade Model. Top: The distribution of the logreturns is heavy-tailed (left) and the logreturns are clustered. Bottom: the autocorrelation functions for the logreturns (left) and absolute logreturns (right) reveal a long memory.

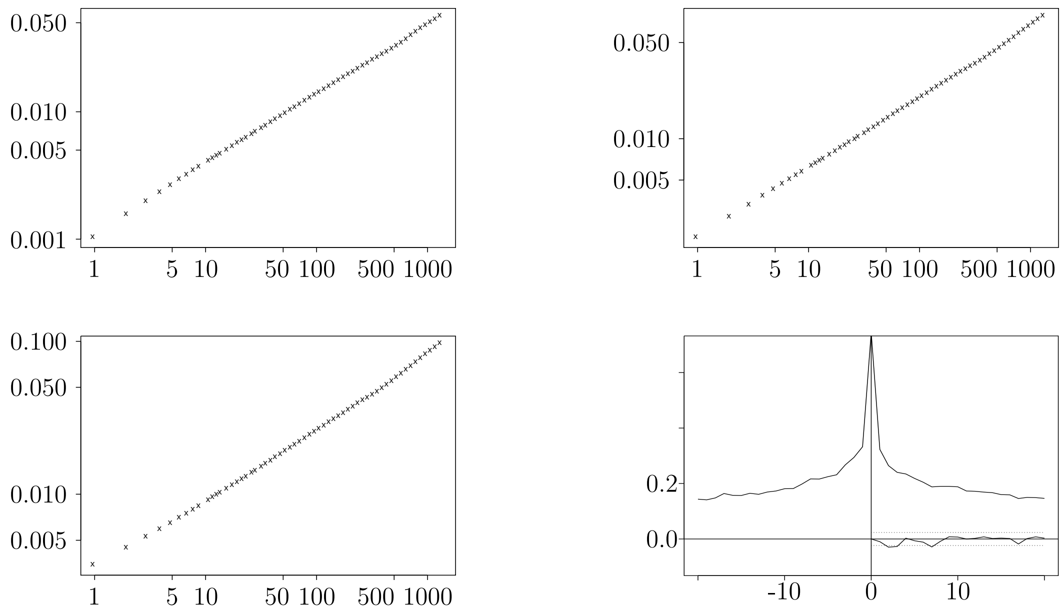

Figure 3.2: The Cascade Model. Top: The scaling laws for $p = 1$ (left) and $p = 2$ (right). Bottom: The scaling law for $p = 3$ (left). There is no asymmetric information flow in the Cascade Model (right).

3.2 Market Microstructure Modeling

Constructing microstructure models has tradition in economics. All these models simulate some aspects of the traders and their behavior. As opposed to time series models, that are designed to capture statistical features in a pure phenomenological way, microstructure models deliver connections between market structures and properties of time series. A survey of microstructure market models is given by Goodhard and O'Hara [20]. LeBaron [21] discussed key design questions and which direction to take when constructing multi agent models. The Heterogeneous Multi Agents Model presented in this thesis is partly inspired by the Minority Game of Challet and Zhang [28] and the Multi Agents Model of Lux and Marchesi [22]. Additionally, we follow the heterogeneous market hypothesis.

The Lux-Marchesi Multi Agents Model

The goal of the Lux-Marchesi Multi Agents Model is to show that heavy tails and volatility clustering arise from the mutual interaction of the market participants. They are divided into two groups of traders. On one side we have the fundamentalists, who expect that the price follows the fundamental value in the long time. They believe in the hypothesis of efficient markets, which assumes that the prices are an immediate and unbiased reflection of incoming news. On the other hand, the noise traders try to identify price trends, which results in a tendency to herding. This group is subdivided into optimists, who always buy, and into pessimists, who always sell. Transition between the different strategy groups is possible and influences the prices. A trader changes his strategy if the promised gain of a new strategy is bigger than of the old one. For the noise traders, the gain is due to price changes, whereas for the fundamentalists, the promised gain corresponds to the difference between the price and the fundamental value. This decision-making process is driven by the exogeneous fundamental value, whose logreturns are Gaussian random numbers. Ie, the fundamental value is not heavy tailed and entails no temporal dependence in volatility.

Most of the time the market is stable. These tranquil phases are interrupted by sudden transient phases of destabilization, ie bursts of volatility. This pattern is known as on-off intermittency in physics and is characterized by an attracting state becoming temporally unstable due to some key variable surpassing a critical value. In the Lux model this parameter is the time-varying fraction of traders pursuing a fundamental and a chartist strategy, respectively. Near the equilibrium, ie when the price and the fundamental value are almost equal, the fraction of agents using one or the other strategy follows a random walk and hence sooner or later leaves this stable area. In this phase of destabilization, a large fraction of the traders switches to chartism, a trend-following strategy, and the price goes up or down. This situation does not last very long since the temporary advantage of chartists disappears when the price bubble breaks down, a situation, where the fundamentalists make higher gains. So the market returns to a stable constellation. This pattern recurs regularly.

The on-off intermittency is the reason for volatility clustering, long memory and heavy tails, all of which the Lux-Marchesi model reproduces.

Minority Game of Zhang

Inspired by the El Farol problem stated by Arthur [1], Challet and Zhang [28] introduced a model for the adaptive evolution of a population of interacting agents, the Minority Game. An odd number N of agents choose at each time step between side 0 or 1. Those agents who have chosen the minority side win and the other loose. In a financial context, the choices 0 or 1 can be replaced by buying or selling an asset. This mechanism can be seen as an abstract version of the law of supply-and-demand. If the majority of traders is buying, it is convenient to be a seller since prices are likely to be high and vice versa. The winners are those on the minority side.

The agent's decision which side to take is based on strategies which process the outcomes of the winning side of the last m time steps and suggest according to this information which side to prefer in the next step. In this binary game, the memory m defines 2^m potential past histories. A strategy is thus represented by a vector S_i $(i = 1, \ldots, 2^m)$ whose elements can be 0 or 1, and the strategy space Γ then corresponds to a hypercube of dimension $d = 2^m$. The total number of strategies is 2^d. At the beginning, each trader draws n strategies from the space Γ. But which strategy of this basket is going to be used? At each time step each strategy gets a virtual point if it has had forecast the winning side. The agent chooses among his n strategies the most successful, ie the one with the most points, and follows its suggestion to choose side 0 or 1. The adaptive nature of the Minority Game lies in the time evolution of the best strategy of each agent. The game has a deterministic time evolution depending only on the initial distribution of the strategies and the memory size m.

An achievement of the Minority Game is that it embodies some basic market mechanisms while keeping the mathematical complexity low and thus allowing for numerical and analytical approaches. Numerical simulations reveal a behavior, where phases of cooperation and herd effects arise, see eg [13] and [11]. A disadvantage of the simple nature of the Minority Game is the absence of a price.

Challet, Chessa, Marsili and Zhang developed a financial market model originating from the Minority Game, see [10] and [12], which is based on the interplay between two types of traders. On one side there are producers, who use the market for exchanging goods. They have a predictable behavior with respect to an exogeneous news arrival process. Producers represent the underlying economic activity. In the absence of other types of traders, the price would be a random walk. On the other side there are speculators, agents with bounded rationality. They study the news arrival process as well as the market's reaction and aim to gain from market fluctuations. They provide liquidity for the producers and are responsible for the generation of stylized facts. Of course, speculators are allowed not to trade when there are no sufficiently profitable opportunities. Transition between equilibrium and out of the equilibrium behavior are observed if the relative number of speculators with respect to the number of producers is changed. This toy market generates heavy-tailed distribution of returns and volatility clustering.

Heterogeneous Multi Agents Model

The goal of the "Heterogeneous Multi Agents Model" we introduce in the next chapter is to reproduce all stylized facts, namely heavy tails, volatility clustering, long memory and especially scaling laws and an asymmetric information flow. In our model we merge the ideas of Lux and Marchesi with those of Challet and Zhang. Ie, we use the ideas of Lux that a trader may switch between technical and fundamental strategies and Zhang's basket of technical strategies. Additionally, we follow the heterogeneous market hypothesis. So our traders act on different time scales, not as in the models of Lux and Zhang. To add some more complexity, every trader can choose between the fundamental strategy and several technical strategies.

4 The Heterogeneous Multi Agents Model

The goal of this thesis was to develop a microstructure model that reproduces all stylized facts discussed in Chapter 2, ie, the logreturn distribution should be heavy-tailed and the volatility should be clustered, have a long memory and obey the scaling laws. Further, the model should yield an asymmetric information flow from long to short time horizons.

In our "Heterogeneous Multi Agents Model" we simulate an FX market, ie, the price of one unit of a foreign currency expressed in units of the domestic currency. The involved traders are acting on different time scales, which reflects the heterogeneous market hypothesis. Every trader has a fundamental strategy and a basket of technical strategies, among which he follows the suggestion of the best one. To apply technical strategies, the traders partly memorize the price history. The traders enter orders consisting of its type (buy or sell), the number of assets (units of the foreign currency) and price (in units of the domestic currency per asset) in an order book, that manages then the trading.

The model is implemented in C++ programming language. The complete source code is given in Appendix C.

4.1 Assumptions of the Heterogeneous Multi Agents Model

The model is based on the following main assumptions, which will be explained in detail in the next chapters.

Heterogeneous Market Hypothesis The market is *heterogeneous*. Traders act on different time scales. In our model they do not switch between them.

Information Hypothesis The *fundamental value* v_t of the foreign exchange rate follows a geometric Brownian motion. Its logreturns are thus normally distributed.

Memory Hypothesis All traders have the *same memory size* to store the latest discretized price changes. The price increments depend on the time scale a trader is acting on.

Strategy Hypothesis Every single trader possesses a *basket of trading strategies*. To each strategy there corresponds a virtual account with wealth $w_t := c_t + a_t\, p_t$, where c_t is the amount of domestic money, a_t the number of assets and p_t is the FX rate at a given time $t = 0, 1, 2, \ldots$. The real account is managed following the suggestion of the strategy corresponding to the virtual account with the largest wealth at time t.

Order Book Hypothesis Traders write an offer, consisting of the type of order, the desired amount of foreign money and the bid- or askprice in an *order book*. An attempt to clear the new order is made immediately after any new entry.

4.2 Heterogeneous Market Hypothesis

In a heterogeneous market, traders act at different frequencies; they have different time horizons. As opposed to a homogeneous market, traders have different reaction times to news or market movements. On the high dealing frequency side we have, for example, market makers, FX traders of banks and day traders, whereas on the low frequency side central banks and pension fund investors are located. In a heterogeneous market, actors are likely to deal at different prices and in different market situations.

In our model, we implement six different time horizons, over which the traders are uniformly distributed. A trader belonging to the group with the shortest time horizon checks the prices and possibly enters an offer in the order book every trading round (1 trading round means contacting all traders). He has a time horizon of one unit in θ-time. A trader on the next higher time scale becomes active every fourth trading round and thus has a time horizon of 4. The other time scales are defined analogously, ie, the time horizons are given by 4^i, where $i = 0, 1, \ldots, 5$ are the names of the time scales.

4.3 Information Hypothesis

Financial markets are exposed to information. We model the information by the *fundamental value* v_t $(t = 0, 1, 2, \ldots)$. This value is supposed to contain all relevant information concerning the FX rate. We assume that the logreturns of v_t at different times are independent normally distributed with mean 0 and variance σ_ϵ^2,

$$\epsilon_t := \log\left(\frac{v_t}{v_{t-1}}\right) \sim N\left(0, \sigma_\epsilon^2\right) \quad (t = 1, 2, \ldots).$$

This means that the fundamental value v_t follows a geometric Brownian motion. The independence and stationarity of the logreturns are important. We are not allowed to build in heavy tails and long memory. These features should arise only from the interaction between the traders. This corresponds to the approach of Lux.

4.4 Memory Hypothesis

We model the fact that every market participant has only a limited information treatment capacity, called *memory size*.

To implement technical strategies, we discretize the price changes, ie, we subdivide price changes over some time interval into five different categories. They represent a weak or strong price increase or decrease or a constant price over a specific time horizon.

For price changes over 4 trading rounds we speak of

a strong price increase (coded as +2) if		$(p_t - p_{t-\Delta t})/p_t$	≥ 0.020,
a weak price increase (+1) if	$0.020 >$	$(p_t - p_{t-\Delta t})/p_t$	≥ 0.005,
a stable price (0) if	$0.005 >$	$(p_t - p_{t-\Delta t})/p_t$	> -0.005,
a weak price decrease (-1) if	$-0.005 \geq$	$(p_t - p_{t-\Delta t})/p_t$	> -0.020,
a strong price decrease (-2) if	$-0.020 \geq$	$(p_t - p_{t-\Delta t})/p_t$,	

where Δt equals 4 trading rounds. To smooth local price changes, p_t is not simply the price at time t, but an exponential moving average of former prices. Exponential moving averages (EMA) are exposed in detail in Zumbach and Müller [29].

Example: If the market turns from a weakly bearish to a strongly bullish one within three time periods, the categorized price process is of the form $\{-1, 0, 2\}$.

Because of computational limitations, we restrict ourselves to a memory size of three. This means that we consider only the last three price increments for a technical strategy to decide whether to buy, hold or sell. We note that for a given memory size of three and for five different price increments there exist 5^3 possible price increment sequences and thus $3^{(5^3)} \approx 4.5 \cdot 10^{49}$ possible strategies.

We stress that the coding of the price process depends on the time horizon of a trader. The chosen criteria for a time interval of 4 trading rounds may be adapted to any other time interval using the *diffusion law*

$$\frac{p_t - p_{t-\Delta t}}{p_t} \approx \sqrt{D\Delta t},$$

where D is the diffusion constant.

Given the chosen criteria for changes over 4 trading rounds, we now calculate the corresponding criteria for changes over $1, 16, 64, 256$ and 1024 trading rounds. We write D_{+2} for the diffusion constant that describes a strong price increase, D_{+1} for the diffusion constant that describes a weak price increase and so on.

For a strong price increase (+2) we get $0.02 := \sqrt{D_{+2}\Delta t}$. Since for 4 trading rounds we define $\Delta t := 1$ we have $D_{+2} = 4 \cdot 10^{-4}$. Analogously we find $D_{+1} = 2.5 \cdot 10^{-5}$. Example

Δt	$+2$	$+1$	-1	-2
1	0.0100	0.0025	-0.0025	-0.0100
4	0.0200	0.0050	-0.0050	-0.0200
16	0.0400	0.0100	-0.0100	-0.0400
64	0.0800	0.0200	-0.0200	-0.0800
256	0.1600	0.0400	-0.0400	-0.1600
1024	0.3200	0.0800	-0.0800	-0.3200

Table 4.1: Criteria for the categorizing of the price increments. Δt is the time horizon in trading rounds. Column "$+2$" lists the barriers a relative return has to be beaten to be called a strong price increase.

to calculate the (+1) barrier of the 64 trading rounds time scale: with $D_{+1} = 2.5 \cdot 10^{-5}$ and $\Delta t = 64/4 = 16$ we get $\sqrt{D_{+1}\Delta t} = 0.02$ as barrier. The criteria for the various time horizons are given in Tab. 4.1.

Example of the coding of price increments: If a trader has a time horizon of 16 trading rounds and if $0.01 < (p_t - p_{t-\Delta t})/p_{t-\Delta t} < 0.04$, he observes a weak price increase (+1).

4.5 Strategy Hypothesis

To make the model as realistic as possible, we allow each trader to choose among different trading strategies.

Real and Virtual Accounts

Every trader has a basket of one real and n virtual accounts, each with wealth

$$w_t = c_t + a_t\, p_t,$$

where c_t is the amount of domestic money, a_t the number of assets and p_t the price of one asset. For each account the initial investments c_0 and a_0 are normally distributed. In particular, short positions are explicitly allowed. But c_0 and a_0 have to suffice the constraint that $w_0 \geq 0$.

If the wealth of the real account is negative over a given time period, the trader is declared to be bankrupt and will be replaced by a new one. See Chapter 4.6 for further details.

Strategies

In addition to the basket of accounts, each trader has a basket of n strategies containing 1 fundamental, 1 optimistic, 1 pessimistic and (n-3) technical strategies.

A *fundamental* strategy is given by

a sell-suggestion if $p_t - v_t \geq 0.005$,

a hold-suggestion if $-0.005 < p_t - v_t < 0.005$ and

a buy-suggestion if $p_t - v_t \leq -0.005$,

where v_t is the fundamental value at time t, as in Chapter 4.3.

The idea is that the price will return to the fundamental value in the long-run. We put 0.005 instead of 0 so that a trader can escape uncertainties in the process of determining the fundamental value v_t.

A *technical* strategy suggestion is based on the past coded price history (see Chapter 4.4). (One technical strategy may suggest to buy after a price process of the form $\{-1, -1, -1\}$, whereas another one may suggest to sell. The technical strategies are initialized in a pure random way and we let the market decide which ones are successful and which ones not.)

An *optimistic* strategy suggests to buy at all costs. This may be considered as a special technical strategy.

A *pessimistic* strategy always suggests to sell, which is a special technical strategy, too.

Strategy Selection

There is a virtual account for each strategy. If active, a trader deals with all n virtual strategies, which has no influence on the FX rate since no orders are placed in the order book. This procedure is nothing but an intern strategy testing. The strategy having the virtual account with the largest wealth is called the *active strategy* and is chosen when the trader becomes active. The trader manages his real account according to the suggestion of his active strategy. This now does have an impact on the FX rate.

4.6 Order Book Hypothesis

Real FX markets are over-the-counter markets, where traders negotiate by phone. In order to keep the programming demands reasonably low, we use an order book as an approximation. An *order* via brokers consists of the type of trade (buy or sell), the number of assets to be traded and the price.

The Composition of the Ask- and Bidprice

If a trader wants to buy or sell, he enters a bid- or askprice, respectively. It seems to be natural that a price offer is based on the following components:

- The *price* p_t, which is simply the FX rate at the time a trader writes his offer.
- The *expected price* e_t, which is the price prediction using exponential moving average operators. They depend on the time scale of the trader.
- The *fundamental value* v_t, which is independent of the time scale.
- The *spread factors* $\exp(\pm X)$, where X has a Gaussian distribution with mean x and variance σ_x^2, that determine the random gap between the ask- and bidprice. The spread, strictly positive and close to 1, will be used as a multiplicative factor. This ensures that its influence does not depend on the scale of the price.

A price offer is given by

$$\begin{aligned} p_{\text{ask}} &= (w_1 p_t + w_2 v_t + w_3 e_t) \exp(+X) \quad \text{and} \\ p_{\text{bid}} &= (w_1 p_t + w_2 v_t + w_3 e_t) \exp(-X), \end{aligned} \tag{9}$$

where the weights $w_1, w_2 \geq 0$ and $w_3 \geq 0$ sum up to 1. The price offer is rounded such that the smallest possible price changes are 1/1000.

Equation (9) ensures that p_{ask} and p_{bid} are always positive.

Trading Restrictions

Usually, the number of assets a trader wants to buy or sell is proportional to his wealth. To make the heterogeneous FX market model as realistic as possible, we impose the following margin requirements.

Sell restriction After a trader has sold all his assets, he is still allowed to sell some more. These short positions are limited to a certain percentage of the amount of money he owns. His money must cover at least eg 5% of the shorted assets.

Buy restriction In analogy to the sell restriction, buying is allowed as long as at least eg 5% of the credit, a negative amount of money, is covered by assets.

Dynamics

A trading round means contacting all N traders in a random sequence. Considering a single trader, we check whether he has had a negative wealth over the last fifty trading rounds. If yes, the trader is bankrupt and gets replaced by a new trader. If not, and given the trader is active, which depends on his time scale, he writes a new order in the order book. Then an attempt to clear this new order is made. If trading is at least partially possible, we get a temporary price $p_{\text{temp},i}$, which equals to the price offer. Otherwise, the order remains in the order book until it gets cleared or overwritten by another order in the same or in the next trading round. This procedure is repeated for all traders. After each trading round we average all temporary prices which results in a new price $p = \overline{p_{\text{temp},i}}$, ie a new FX rate. The number of repetitions of this procedure, ie the number of trading rounds, determines the length of the price time series.

The fundamental value v_t, the spread X, the choice of initial strategies and the initial wealth are random inputs. Also the strategies are randomly initialized.

4.7 Implementation of the Heterogeneous Multi Agents Model

The “Heterogeneous Multi Agents Model” is implemented in C++ programming language. The complete source code is given in Appendix C.

We give an overview of which aspect of the model is implemented in which class, respectively which file(s).

- The heterogeneous market hypothesis, ie the grid of the different time scales is implemented in the class *Mmc* declared in the files “Mmc.h” and “Mmc.cpp”.

- The information flow hypothesis is implemented in the class *Information* declared in the file “Information.h”.

- Exponential moving averages (EMA) are implemented in the class *Ema* declared in the file “Ema.h”.

- All aspects of coding a price process (the memory hypothesis) are implemented in the class *CodedPrice*, declared and defined in the files “CodedPrice.h” and “CodedPrice.cpp”, respectively.

- The strategy hypothesis, ie accounts and strategies are implemented in the class *Portfolio* declared and implemented in the files “Portfolio.h” and “Portfolio.cpp”.

- The classes *Trader* and *OrderBook*, declared and implemented in the files “Trader.h”, “Trader.cpp”, “OrderBook.h” and “OrderBook.cpp” treat all features of the market mechanism (the order book hypothesis).

5 Simulation Runs and Discussion

In this chapter we show that there exists a parameter set of the "Heterogeneous Multi Agents Model" that reproduces the stylized facts not only in a qualitative way, but also replicates the tail index α, the characterizing parameter of the autocorrelation function of the volatility h and the scaling exponents D_1, D_2 and D_3 such that they do not significantly differ from real FX data. This parameter set is called *default parameter set* in the following and is introduced in Chapter 5.1. The model does also generate an asymmetric information flow. In Chapter 5.2 we show that the default parameter set generates all stylized facts and comparisons with the real FX price time series presented in Chapter 2 are made. Many simulations are made to allow for a thorough statistical discussion. To test the stability of the model, we perform a stress-testing as described in Chapter 5.3. Starting with the default set, we slightly change one parameter and examine the consequences. Chapter 5.4 shows the necessity of following the heterogeneous market hypothesis.

5.1 Default Parameter Set

One very successful parameter set, the default parameter set, is characterized by 1500 traders, acting on 6 different time scales. The corresponding time horizons are given by powers of 4. Ie traders acting on the lowest time scale have a horizon of 1; they are active in every trading round. On the next higher time scale, traders act every 4th trading round, on the third scale every 16th round and so on. Every trader is equipped with 10 strategies (1 fundamental, 1 optimistic, 1 pessimistic and 7 technical strategies, as described in Chapter 4.5). When writing an offer in the order book, the price offer is, as discussed in Chapter 4.6, given by

$$
\begin{aligned}
p_{\text{ask}} &= (w_1 p_t + w_2 v_t + w_3 e_t) \exp(+X) \quad \text{and} \\
p_{\text{bid}} &= (w_1 p_t + w_2 v_t + w_3 e_t) \exp(-X),
\end{aligned}
$$

where p_t is the latest observed price, v_t the fundamental value and e_t the price prediction. We choose the weights in the following way: $w_1 = 0.4, w_2 = 0.3$ and $w_3 = 0.3$. They are chosen to be equal for all traders, regardless of their time scale. X, characterizing the price spread, has a Gaussian distribution with mean 0.0002 and standard deviation 0.0005. The fundamental value, the driving process of the market, is characterized by the standard deviation of its logreturns, which is chosen to be 0.001. A run consists of 100000 trading rounds. The default parameter set is summarized in Tab. 5.1.

The "S-Plus" script written to analyze the price time series is listed in Appendix B. It contains tests for the heavy tails, long memory, scaling laws and the asymmetric information flow.

parameter name	value	name in S-Plus script
number of traders	1500	NbTraders
number of strategies	10	NbStrategies
initial fundamental value	1.0	InitFundValue
initial FX rate	1.0	InitMeanPrice
default order type	hold	DefaultOrderType
mean half price spread	0.0002	PriceSpreadHalf
std dev of half price spread	0.0005	PriceSpreadStdDev
std dev of fundamental value	0.001	FundValueStdDev
wealth weight for trading	0.2	WealthTradingWeight
credit limit	0.05	CreditLimit
price weight w_1	0.4	PriceWeight1
price weight w_2	0.3	PriceWeight2
price weight w_3	0.3	PriceWeight3

Table 5.1: The parameters of the default set. The price weights w_1, w_2 and w_3 are equal for all time scales. The corresponding S-Plus script is listed in Appendix B.

5.2 Comparison with Empirical Results

Typical Runs

We show that the "Heterogeneous Multi Agents Model" together with the default parameter set reproduces the stylized facts. Four typical runs are given in the following and have to be compared with the plots of real FX rates we discussed in Chapter 2.

In Fig. 5.1 we see that the price process looks quite natural. It has various structures and substructures and the overall price movements are not too extreme, which is expected for real major FX rates (see Fig. 2.1). The five-figure numbers left of the y-axis denote the random seed of the run and are used to identify the plot.

Looking at the logreturns in Fig. 5.2, we observe clusters of volatility, as we do in Fig. 2.2, which shows real logreturns.

A consequence of the volatility clustering are heavy tails in the distribution of the logreturns, see Fig. 5.3.

From Fig. 5.4 (compare with Fig. 2.3) we conclude that the autocorrelation function (acf) of the logreturns is mostly within the 95% confidence interval of white noise. Hence, the logreturns are uncorrelated except for the first few lags.

Examining absolute logreturns, the volatility, the behavior of the acf is different than in the case of logreturns. In Fig. 5.5 (compare with Fig. 2.4) the acf is greater than 0 for a large

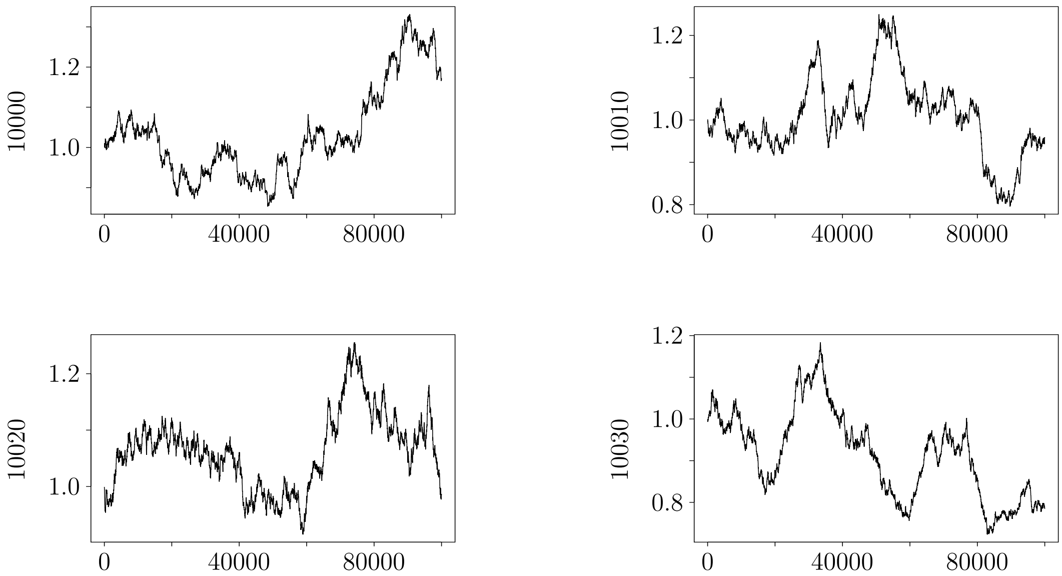

Figure 5.1: The price processes of the "Heterogeneous Multi Agents Model" reveal various structures and substructures, as real FX rates do. The five-figure numbers on the y-axis denote the random seed of the run.

number of lags, which indicates a long memory. This is consistent with the observation of volatility clustering. The market remembers whether it was quiet or in an uproar in the near past. The "Heterogeneous Multi Agents Model" has, as real FX markets, a long memory.

In Figs. 5.6, 5.7 and 5.8 we can see that the scaling laws, described in Chapter 2.5, do hold for $p = 1, p = 2$ and $p = 3$ because the log-log plots reveal straight lines. Our "Heterogeneous Multi Agents Model" is the first microstructure model that generates the scaling laws correctly.

The "Heterogeneous Multi Agents Model" is also the first microstructure model that is able to generate an asymmetric information flow. This can be seen in Fig. 5.9, which shows similar plots as Fig. 2.9 does for real FX price time series. The correlation of past coarse and future fine volatility is plotted in the left branch of the upper curve and the correlation of past fine and future coarse volatility is plotted in the right branch of the upper curve. The difference between the left and the right branch is shown by the lower curve and lies, for the first few lags, outside the 95% confidence interval of white noise, which is indicated by the dotted lines.

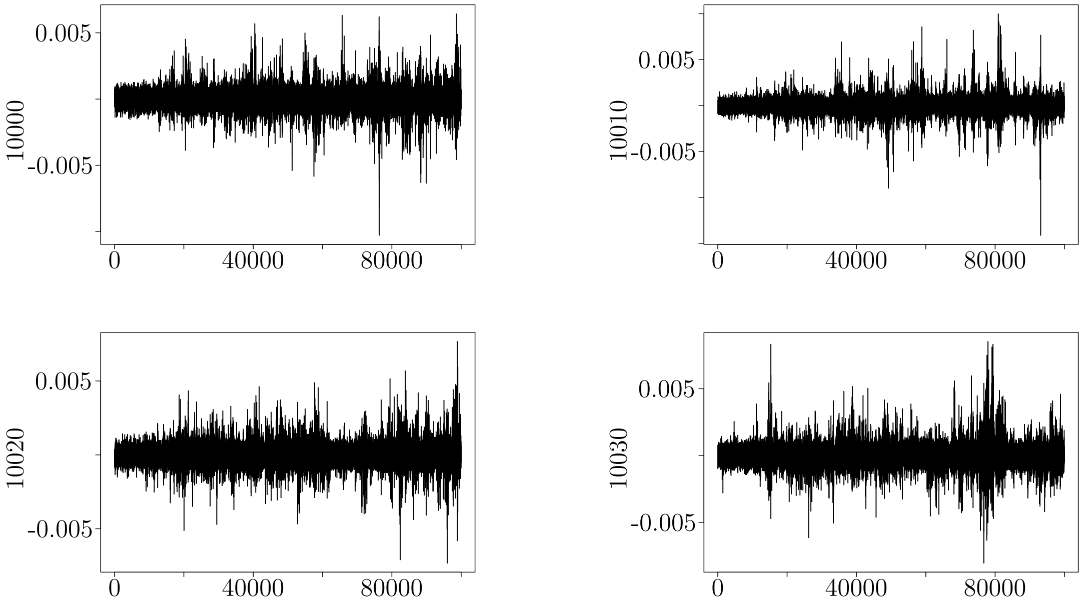

Figure 5.2: The logreturns of the "Heterogeneous Multi Agents Model" are clustered.

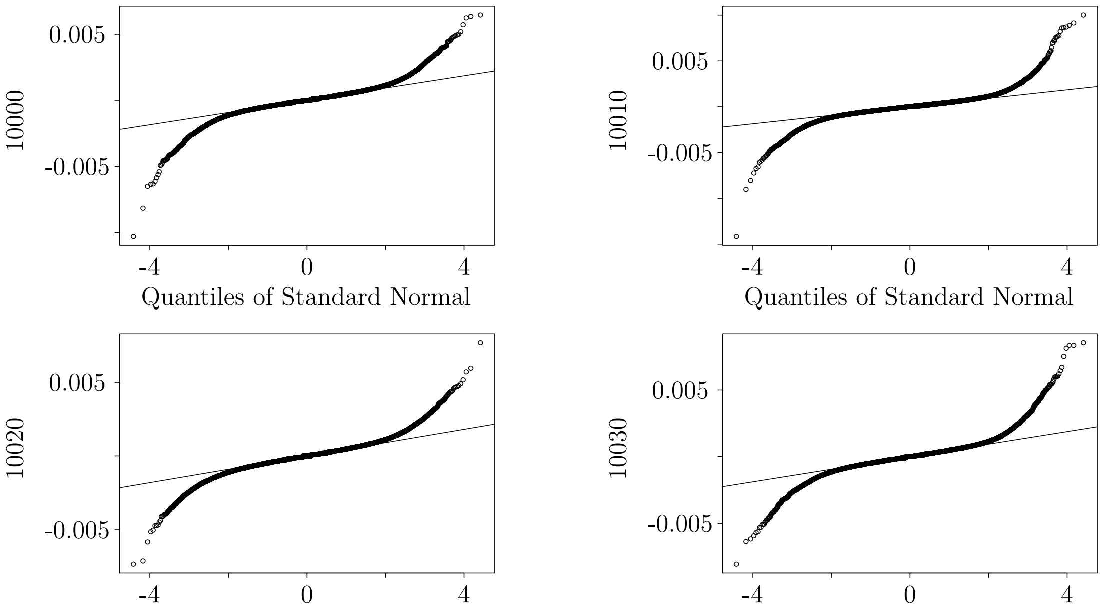

Figure 5.3: The distribution of the logreturns of the "Heterogeneous Multi Agents Model" is heavy-tailed.

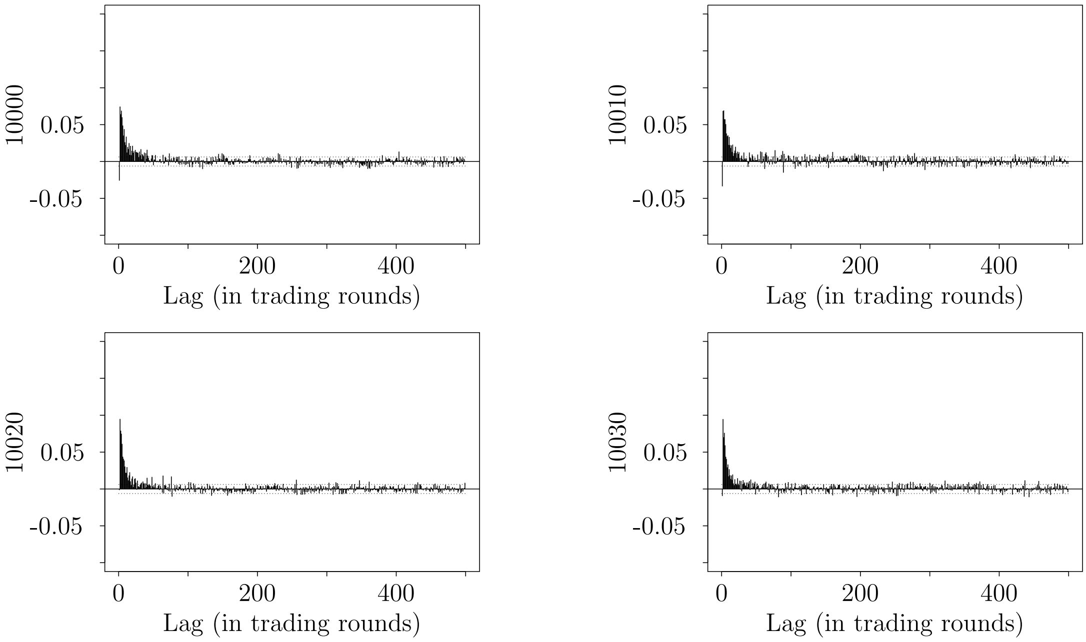

Figure 5.4: The logreturns of the "Heterogeneous Multi Agents Model" are, aside from the first few lags, uncorrelated. The value of the autocorrelation function corresponding to lag 0 is not plotted.

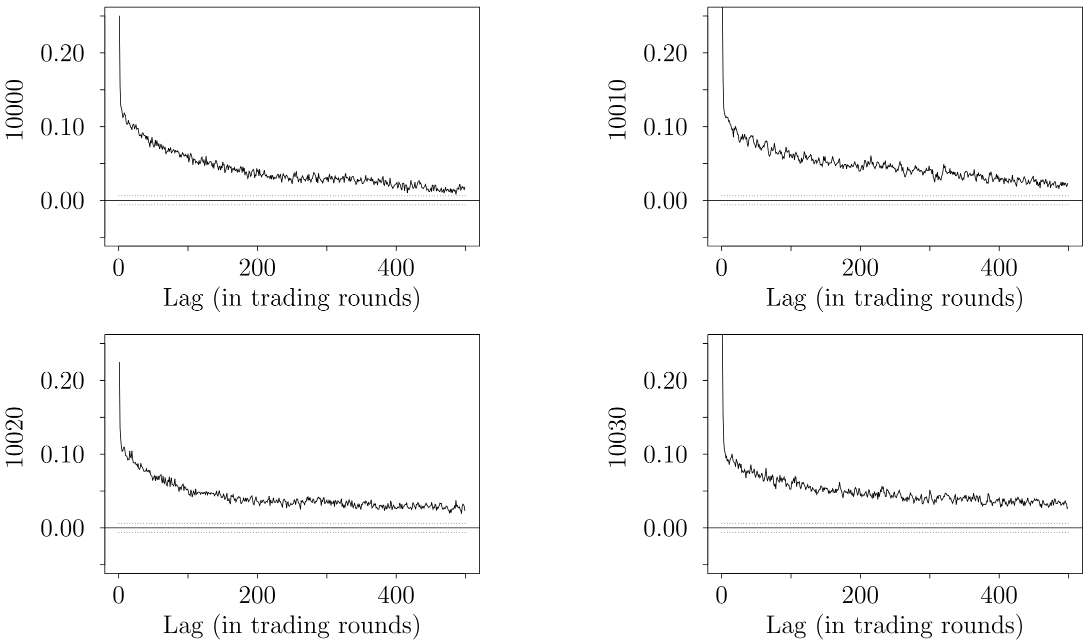

Figure 5.5: The volatility of the "Heterogeneous Multi Agents Model" has a long memory. The value of the autocorrelation function corresponding to lag 0 is not plotted.

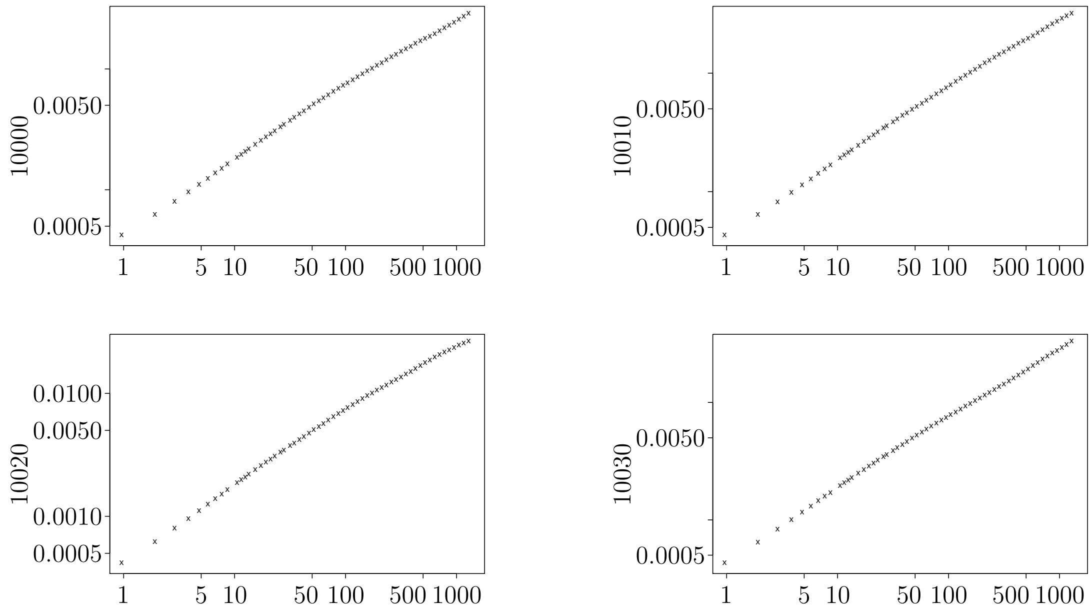

Figure 5.6: The "Heterogeneous Multi Agents Model" produces the correct scaling law for $p = 1$. Not all points are plotted.

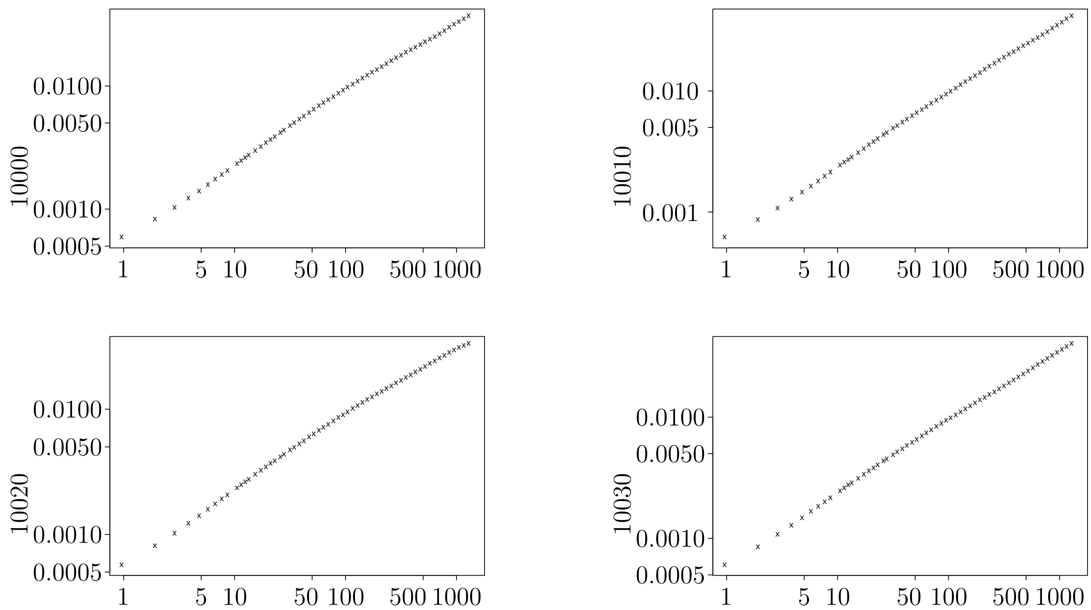

Figure 5.7: The "Heterogeneous Multi Agents Model" produces the correct scaling law for $p = 2$. Not all points are plotted.

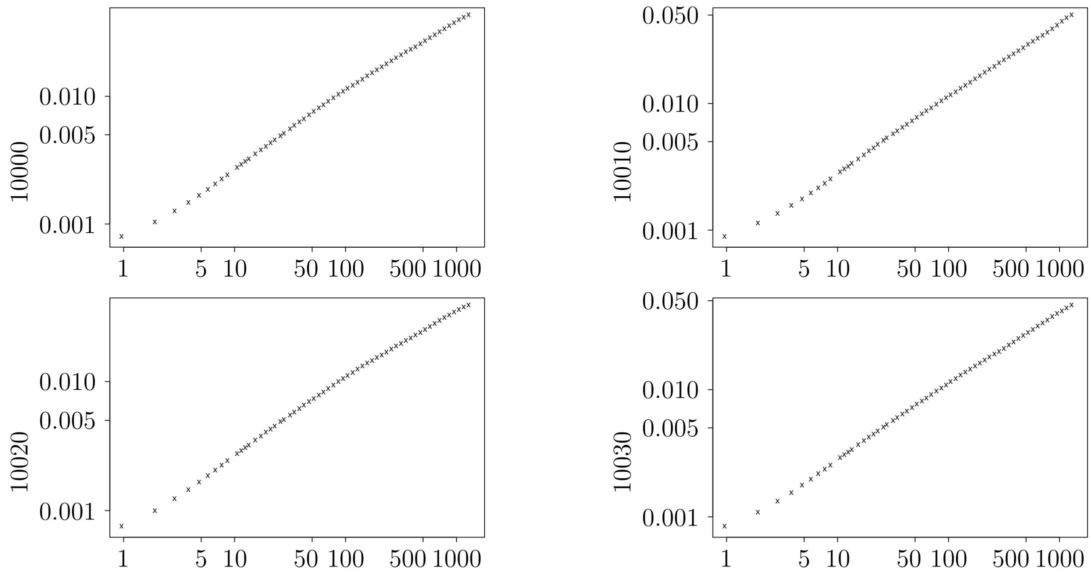

Figure 5.8: The “Heterogeneous Multi Agents Model” produces the correct scaling law for $p = 3$. Not all points are plotted.

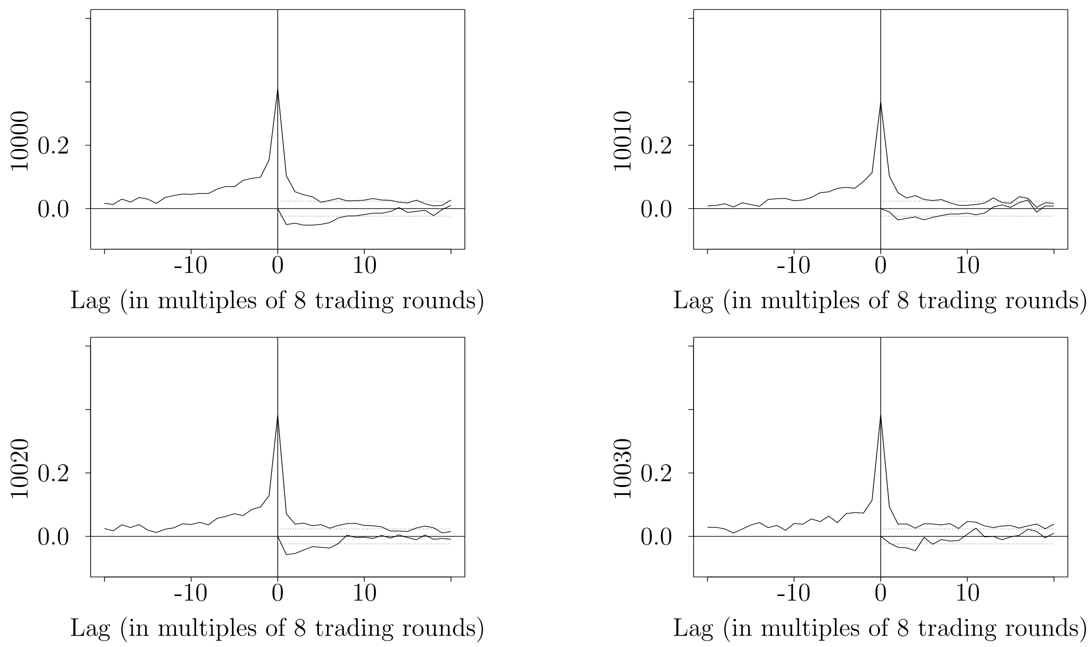

Figure 5.9: Asymmetric lagged correlation function of fine and coarse volatilities. In the left branch of the upper curve, the correlation of past coarse and future fine volatility is plotted and vice versa in the right branch. The lower curve, representing the difference of the left and right branch, indicates the asymmetry. Coarse volatility predicts fine volatility better than vice versa. The dotted lines are the 95% confidence interval of white noise.

Statistical Tests

In order to minimize random effects while estimating the characterizing parameters of the stylized facts, namely the tail index α, n and h describing the long memory, the scaling exponents D_1, D_2 and D_3 and the difference of the lagged correlation functions, we make 30 simulation runs à 100000 trading rounds (ie, each time series contains 100000 price observations) and then determine the means and confidence intervals. (Because the simulation of the price process and its analysis is quite time consuming, we restricted ourselves to 30 runs. However, the next computer generation will allow for a better statistics, ie, more runs.)

Long Memory We test whether the volatility has a long memory. To do so, we follow the procedure described in Chapter 2.3 and in a first step estimate n. Tab. 5.2 clearly suggests that the volatility has a long memory, this because 0 is not in the 95% confidence interval of n. (This 95% confidence interval is given by $[mean(n) - 2 * stddev(n)/\sqrt{30}, mean(n) + 2 * stddev(n)/\sqrt{30}]$. Admittedly, 30 values are not quite enough to use the central limit theorem. We therefore put 2 instead of 1.96 to be on the safe side.) We fit the acf to the hyperbolical model given in (3), namely $\rho_h(\tau) = c\tau^{-h}$. The corresponding estimates of h are given in Tab. 5.2, too.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
n	2.40	2.85	3.11	3.08	3.31	3.59	[2.96, 3.20]
h	0.271	0.294	0.313	0.319	0.340	0.399	[0.307, 0.332]

Table 5.2: Estimation of n and h for the default parameter set.

When comparing the simulated values for n and h with the estimated values of real FX rates in Tab. 2.1, we see that the mean of h is at the upper end of the observed values. This means that the autocorrelation of the simulated time series is slightly smaller than for real data.

Heavy Tails To check whether the distribution of the logreturns has heavy tails, we estimate the shape parameter and thus the tail index of the POT model, discussed in Chapter 2.4, using the 5000 largest logreturns. In Tab. 5.3 we see that α is approximately 4.5, which is in the range of the estimated tail indices of real data, where the logreturns are measured over 1 hr θ-time (see Tab. 2.2).

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
α	3.45	3.96	4.40	4.53	5.03	6.14	[4.26,4.79]

Table 5.3: Estimation of the tail index α for the default parameter set.

Scaling Laws We estimate the scaling exponents D_p of (5) for $p = 1, p = 2$ and $p = 3$

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
D_1	0.517	0.539	0.550	0.554	0.569	0.588	[0.546,0.561]
D_2	0.519	0.534	0.549	0.552	0.568	0.592	[0.544,0.559]
D_3	0.511	0.530	0.548	0.549	0.565	0.598	[0.541,0.558]

Table 5.4: Estimation of the scaling exponents D_1, D_2 and D_3 for the default parameter set.

stated in Chapter 2.5. Tab. 5.4 shows that $D_1 \approx 0.554, D_2 \approx 0.552$ and $D_3 \approx 0.549$. Real FX data, where the logreturns are measured over 1 hr θ-time are at about 0.53 for D_1, 0.51 for D_2 and 0.50 for D_3, see Tab. 2.3. Thus the modeled value for D_1 is quite good. We also have $D_1 > D_2 > D_3$, but not as clear as in the case of real data.

Asymmetric Information Flow Tab. 5.5 shows the difference of the lagged correlations of the first 9 lags measured in multiples of 8 trading rounds. This difference is the mean of the 30 simulation runs. Comparing to the 95% confidence interval of white noise given by $[-0.0237, +0.0235]$ (as discussed in Chapter 2.6), we conclude that there is a significant information flow from long to short time horizons for the first 5 lags. The information flow of the "Heterogeneous Multi Agents Model" is similar to the information flow in real FX time series, as can be seen in Tab. 2.4.

lag 1	lag 2	lag 3	lag 4	lag 5	lag 6	lag 7	lag 8	lag 9
-0.0288	-0.0412	-0.0342	-0.0313	-0.0243	-0.0208	-0.0192	-0.0172	-0.0138

Table 5.5: Mean difference of lagged correlations of the first 9 lags (in multiples of 8 trading rounds) for the default parameter set. The negative sign means that coarse volatility predicts fine one better than vice versa. The 95% confidence interval for white noise is $[-0.0237, +0.0235]$.

5.3 Stress Testing

The parameter space of the "Heterogeneous Multi Agents Model" has many dimensions. It is therefore impossible to examine every possible parameter configuration with the computer technology standing at our disposal.

To get an idea of how stable the model is, we perform a so-called *stress testing*. Starting from the default parameter set, we vary a single model parameter and examine how the parameters characterizing the stylized facts change. More exactly, one after the other model parameter is slightly increased or decreased. We run every modified parameter set 30 times à 100000 trading rounds and with the same seeds of the random generators as for the default parameter set to make a comparison possible. We then perform the same statistical analysis as for the default parameter set, ie, we examine the heavy tails, represented by α, the long memory represented by n and h as well as the scaling laws with the characteristic scaling exponents D_1, D_2 and D_3.

The names of the modified parameter sets and the corresponding modified values are given in Tab. 5.6 and the complete results of all modified parameter sets are listed in Tab. A.1 to Tab. A.6 in Appendix A.

name of parameter set	modified value
1000 traders	number of traders = 1000
2000 traders	number of traders = 2000
5 strategies	number of strategies = 5
20 strategies	number of strategies = 20
buy order	default order type = buy
sell order	default order type = sell
0 price spread	half price spread: mean = 0 and std dev = 0
1 price spread	half price spread: mean = 0.0003 and std dev = 0.0007
15 fund value	std dev of fundamental value = 0.0015
075 fund value	std dev of fundamental value = 0.00075
1 wealth weight	wealth weight for trading = 0.1
3 wealth weight	wealth weight for trading = 0.3
01 credit limit	credit limit = 0.01
5 credit limit	credit limit = 0.5
334 price weights	$w_1 = 0.3, w_2 = 0.3, w_3 = 0.4$
343 price weights	$w_1 = 0.3, w_2 = 0.4, w_3 = 0.3$
424 price weights	$w_1 = 0.4, w_2 = 0.2, w_3 = 0.4$
43525 price weights	$w_1 = 0.4, w_2 = 0.35, w_3 = 0.25$
523 price weights	$w_1 = 0.5, w_2 = 0.2, w_3 = 0.3$
52525 price weights	$w_1 = 0.5, w_2 = 0.25, w_3 = 0.25$

Table 5.6: The names of the various parameter sets used for the stress testing together with the only parameter changed with respect to the default parameter set. The price weights w_1, w_2, w_3 are equal for all time scales.

In Tab. 5.7, a summary of Tab. A.1 to Tab. A.6, we give an overview of significant changes of stylized facts parameters in the modified sets with respect to the default parameter set. A " + " sign indicates that the mean of this parameter is above the confidence interval of the default set, whereas a " − " sign means that the value lies below. A "0" indicates no significant changes.

	h	α	D_1	D_2	D_3
01 credit limit	0	0	0	0	0
5 credit limit	0	+	0	0	0
buy order	0	0	0	0	0
sell order	0	0	0	0	0
15 fund value	+	0	0	0	0
075 fund value	−	+	0	0	0
5 strategies	0	0	0	0	0
20 strategies	+	0	0	0	0
0 price spread	+	0	0	0	0
1 price spread	−	0	0	0	0
334 price weights	0	0	0	0	0
343 price weights	0	0	−	0	0
424 price weights	−	0	+	+	+
43525 price weights	+	+	−	−	−
523 price weights	0	0	+	+	+
52525 price weights	+	0	+	+	+
1000 traders	+	0	0	0	0
2000 traders	0	+	0	0	0
1 wealth weight	0	0	0	0	0
3 wealth weight	0	0	0	0	0

Table 5.7: Significant changes of parameters characterizing the stylized facts with respect to the default parameter set. A " + " (" − ") sign indicates an increase (decrease) of the mean with respect to the mean of the default parameter set, given that it is outside of the confidence interval.

It is difficult to interpret Tab. 5.7 since the mutual dependencies within the parameters of the "Heterogeneous Multi Agents Model" are not trivial. We make some statements below:

Let us start with the long memory parameters h and n.

- The fundamental value is an ingredient of the price offers. A big volatility of the fundamental value implies a big volatility of the price and thus results in a shorter memory.

- As intuitively expected, h is inversely related to the number of traders; more traders means longer memory.
- If the bid-ask spread vanishes, the market is more liquid and an equilibrium is faster reached, so the memory becomes shorter.
- It is worth mentioning that all modified parameter sets do have a long memory because the confidence interval of n never includes 0, as can be seen in Tab. A.2.

Looking at the tail index α, we see that more traders involved results in more price offers, which smooths the price process and thus leads to an increase of the tail index, respectively to a decrease in the probability of extreme events.

Interestingly enough, only the price weights w_1, w_2 and w_3 seem to have a significant influence on the scaling exponents D_1, D_2 and D_3.

In Tab. 5.8 we see that all modified parameter sets are very robust with respect to the shifts of the parameters in lagged correlations. Sometimes lag 1 lies not outside the confidence interval of white noise, but the next few lags definitely do. This information flow from long to short horizons is therefore well established in the "Heterogeneous Multi Agents Model".

The goal to build the "Heterogeneous Multi Agents Model" as realistic as possible has the disadvantage that it became a little bit too complicated. The stress testing reveals the weak impact of trading restrictions (parameter: credit limit) and of the trading weight of a trader (parameter: wealth weight) on the stylized facts. In a future revision of the "Heterogeneous Multi Agents Model", these two parameters may perhaps be omitted.

	lag 1	lag 2	lag 3	lag 4	lag 5	lag 6	lag 7
default set	-0.0288	-0.0412	-0.0342	-0.0313	-0.0243	-0.0208	-0.0192
01 credit limit	-0.0302	-0.0408	-0.0376	-0.0331	-0.0255	-0.0248	-0.0227
5 credit limit	-0.0269	-0.0424	-0.0391	-0.0324	-0.0299	-0.0257	-0.0223
buy order	-0.0276	-0.0436	-0.0378	-0.0320	-0.0296	-0.0234	-0.0192
sell order	-0.0291	-0.0477	-0.0398	-0.0341	-0.0277	-0.0249	-0.0204
15 fund value	-0.0340	-0.0496	-0.0414	-0.0350	-0.0306	-0.0236	-0.0229
075 fund value	-0.0249	-0.0402	-0.0354	-0.0279	-0.0238	-0.0243	-0.0203
5 strategies	-0.0290	-0.0413	-0.0392	-0.0307	-0.0287	-0.0240	-0.0242
20 strategies	-0.0349	-0.0466	-0.0393	-0.0301	-0.0248	-0.0217	-0.0179
0 price spread	-0.0256	-0.0462	-0.0386	-0.0331	-0.0300	-0.0254	-0.0234
1 price spread	-0.0276	-0.0402	-0.0368	-0.0286	-0.0271	-0.0216	-0.0185
334 price weights	-0.0289	-0.0480	-0.0412	-0.0381	-0.0274	-0.0219	-0.0171
343 price weights	-0.0285	-0.0386	-0.0336	-0.0270	-0.0210	-0.0168	-0.0159
424 price weights	-0.0053	-0.0455	-0.0368	-0.0341	-0.0334	-0.0296	-0.0201
43525 price weights	-0.0345	-0.0416	-0.0300	-0.0236	-0.0262	-0.0226	-0.0199
523 price weights	-0.0114	-0.0441	-0.0421	-0.0374	-0.0360	-0.0296	-0.0203
52525 price weights	-0.0263	-0.0477	-0.0389	-0.0355	-0.0339	-0.0262	-0.0238
1000 traders	-0.0222	-0.0431	-0.0394	-0.0294	-0.0243	-0.0216	-0.0224
2000 traders	-0.0291	-0.0450	-0.0351	-0.0330	-0.0272	-0.0254	-0.0211
1 wealth weight	-0.0336	-0.0464	-0.0432	-0.0353	-0.0322	-0.0293	-0.0238
3 wealth weight	-0.0280	-0.0426	-0.0353	-0.0314	-0.0268	-0.0249	-0.0197

Table 5.8: Mean difference of lagged correlations of the first 7 lags (in multiples of 8 trading rounds) for all modified default sets. The negative sign means that coarse volatility predicts fine one better than vice versa. The 95% confidence interval for white noise is $[-0.0237, +0.0235]$.

5.4 Test of the Heterogeneous Market Hypothesis

We once more emphasize that it is very important that the traders act on different time scales. If this heterogeneity is no longer given, several stylized facts disappear as can be seen in Fig. 5.10 and Fig. 5.11. In our “Heterogeneous Multi Agents Model” all stylized facts, except the scaling laws, disappear when the traders act on a single time scale. We give the example where the time horizon is chosen to be 4 trading rounds and where all other values of the default parameter set remain unchanged. Of course, this parameter configuration no longer deserves the adjective heterogeneous.

In the case of two different time scales, we choose 4 and 64 trading rounds as time horizons as an illustration, some of the stylized facts arise again, even so not very convincing. In Fig. 5.12 we see that the price process looks very natural and that the distribution of the logreturns is heavy-tailed. But there is no volatility clustering and no long memory. Fig. 5.13 shows scaling laws and an asymmetric information flow, stronger than in real data.

However, it is not sufficient just to implement 6 different time scales. They also have to be chosen carefully. If not, some stylized facts may disappear as, eg, in the case where the time scales are powers of 2 trading rounds (see Figs. 5.14 and 5.15). In this case, the collective memory of the traders can not reach far enough in the past. In Fig. 5.14 we see that the autocorrelation function of the volatility is outside of the confidence interval of white noise only up to a lag of about $2^5 = 32$.

It is necessary to follow the hypothesis of a heterogeneous market and to choose the time scales carefully in order to generate all stylized facts.

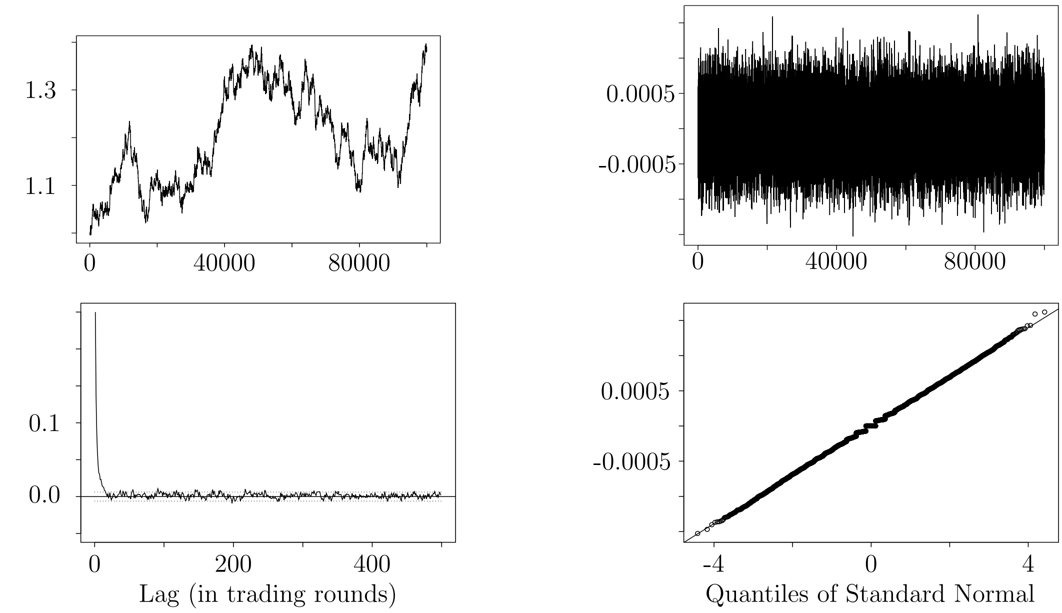

Figure 5.10: If all traders act on one single time scale, the “Heterogeneous Multi Agents Model” is actually no longer heterogeneous and the volatility clustering, the long memory and the heavy tails disappear, although the price process looks still natural.

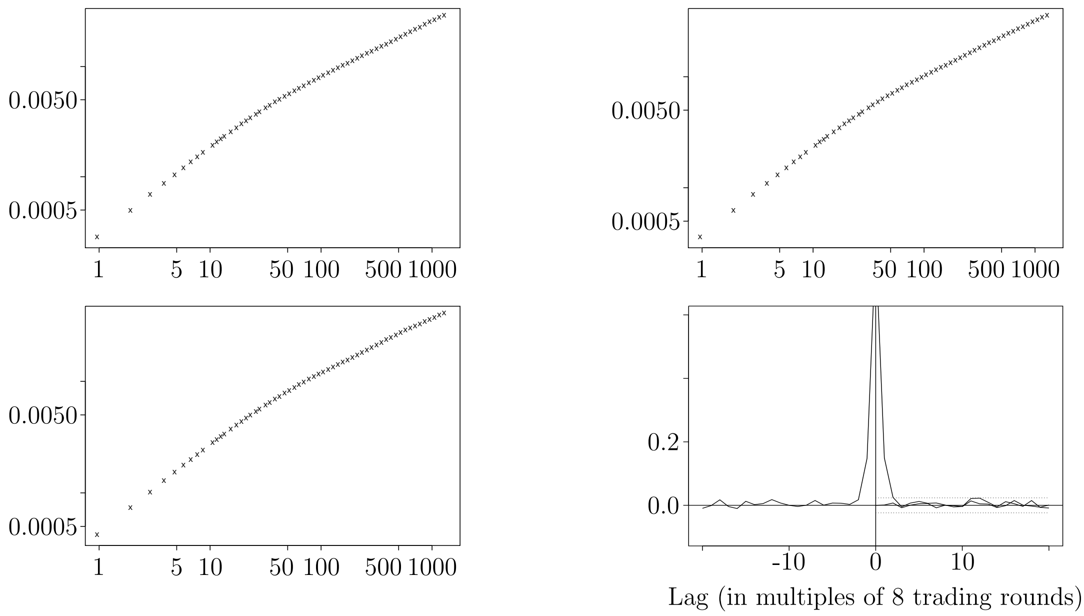

Figure 5.11: If all traders act on one single time scale, the “Heterogeneous Multi Agents Model” is actually no longer heterogeneous. The scaling laws are less convincing and there is no longer an asymmetric information flow.

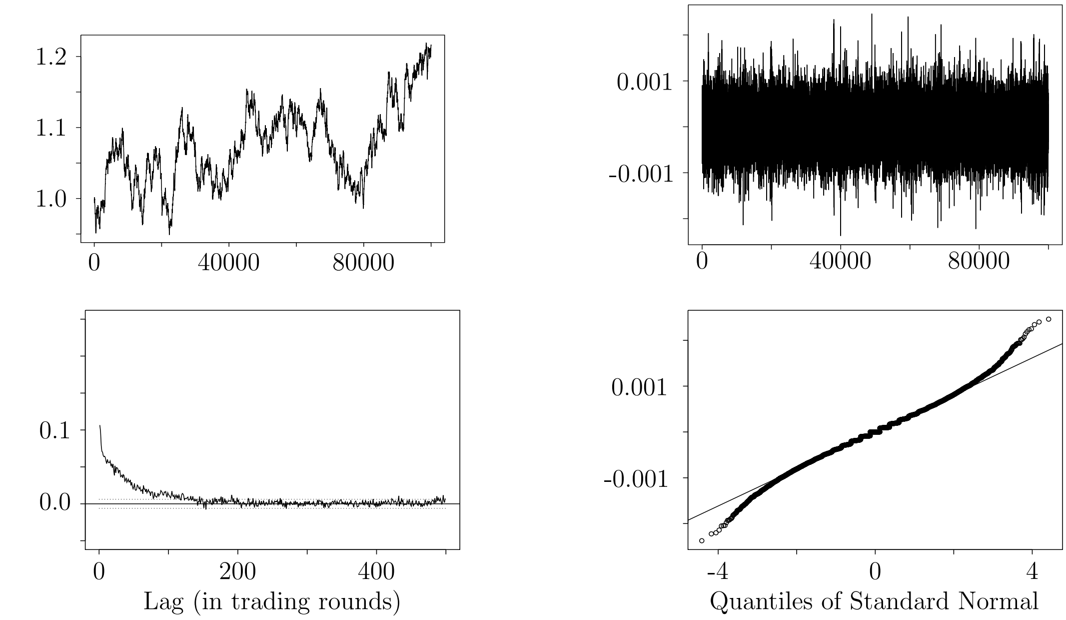

Figure 5.12: If all traders act on only two different time scales, the “Heterogeneous Multi Agents Model” generates no volatility clustering and has no long memory. The distribution of the logreturns, however, is heavy-tailed.

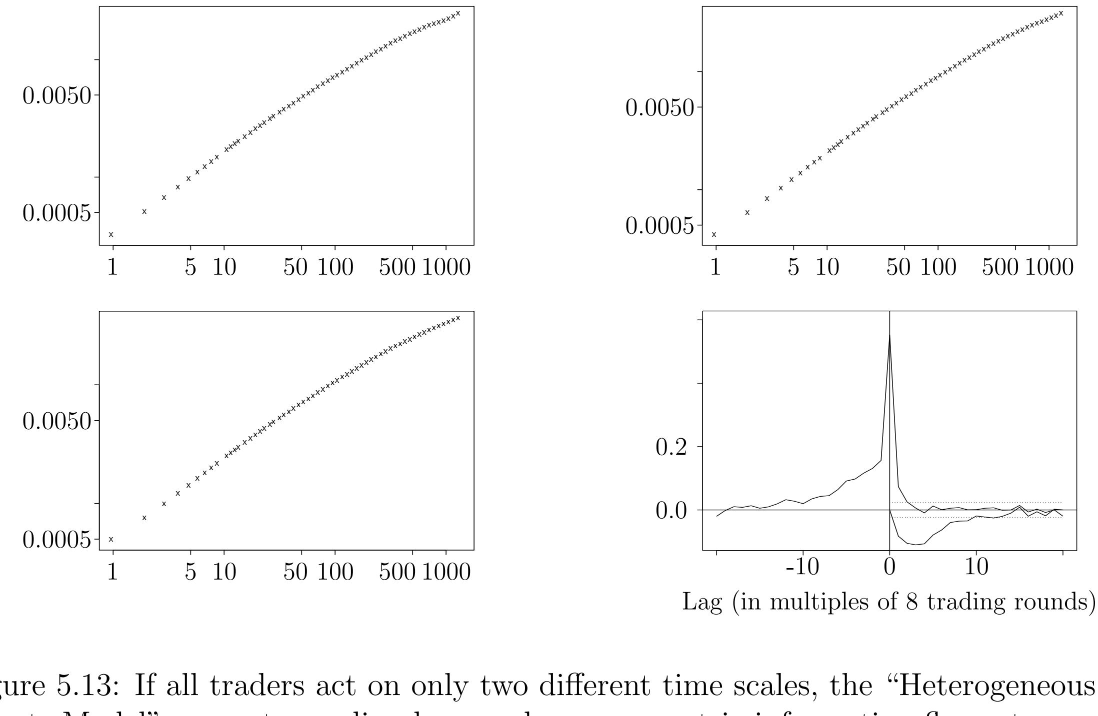

Figure 5.13: If all traders act on only two different time scales, the “Heterogeneous Multi Agents Model” generates scaling laws and an asymmetric information flow, stronger than in real data.

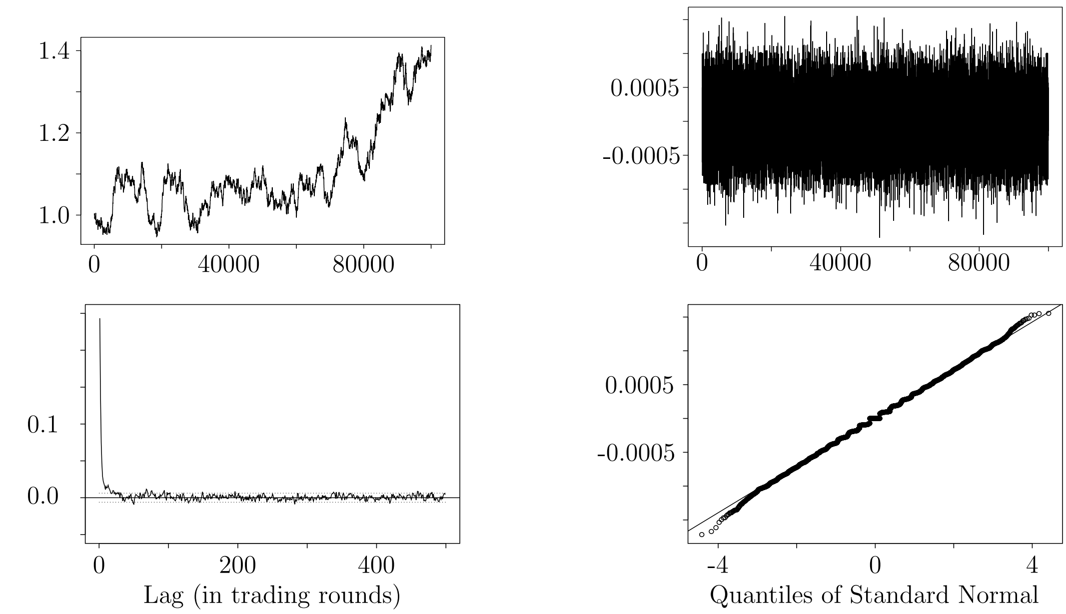

Figure 5.14: If the traders act on six unsuitably chosen time scales, the "Heterogeneous Multi Agents Model" generates no volatility clustering, has no long memory and the distribution of the logreturns is not heavy-tailed.

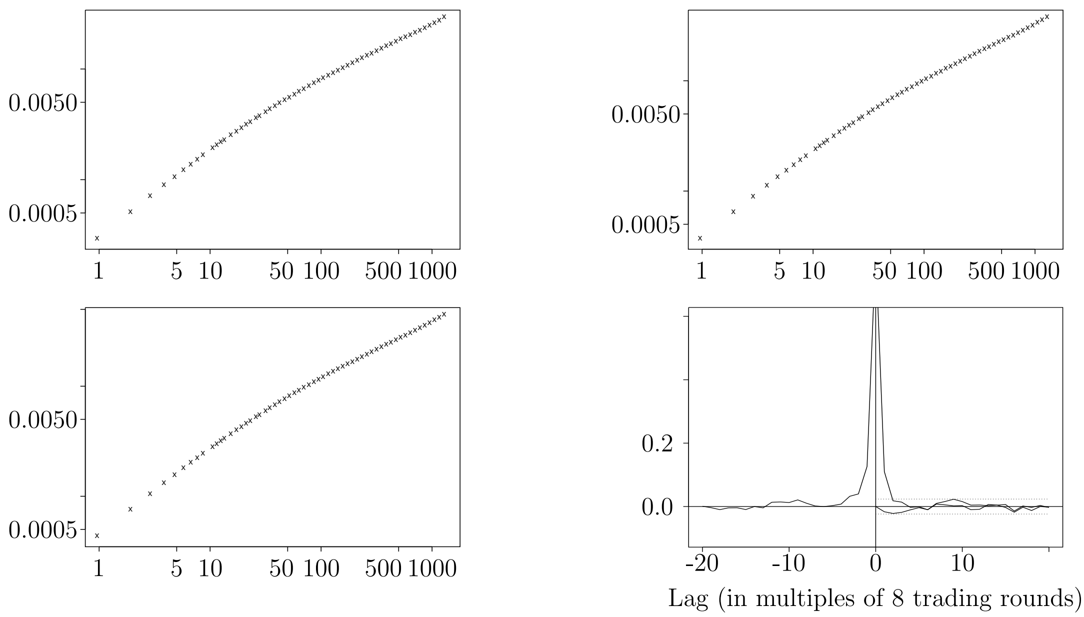

Figure 5.15: If the traders act on six unsuitably chosen time scales, the "Heterogeneous Multi Agents Model" does still generate scaling laws, but exhibits no longer an asymmetric information flow.

6 Summary and Outlook

In this thesis - using the FX rates EUR/CHF, EUR/USD, USD/CHF and USD/JPY - we illustrated the well-known fact that financial time series share certain stylized facts, namely that the distribution of the logreturns is heavy-tailed and that the volatility is clustered and has a long memory. Further, the volatility measured over different time intervals reveals certain scaling laws. There is also an information flow from long to short time horizons. It is known that these statistical properties are shared by other FX rates and stock markets, too.

In order to explain these observed facts we built the "Heterogeneous Multi Agents Model", which is a microstructure model with agents acting on different, carefully chosen time scales. These traders are equipped with a basket of strategies among which they follow the suggestion of the most successful one and enter their buy or sell offers in an order book, that manages the trading. Design decisions were made following the idea "the more realistic the model is, the more stylized facts it will generate".

We found a default parameter set of the "Heterogeneous Multi Agents Model" that generates the stylized facts not only in a qualitative way, but that can also reproduce the characterizing parameters of the stylized facts of real FX data, ie h of the long memory, the tail index α and the scaling exponents D_1, D_2 and D_3. A stress-testing confirms the stability of the model. We have also seen the importance that the traders act on more than just one or two different time scales and that even if there are six time scales, we have to choose them suitably. The time horizons should be chosen in an equidistant way on a logarithmic scale and should cover a sufficiently large range. The "Heterogeneous Multi Agents Model" delivers a strong hint for the heterogeneous market hypothesis, ie the behavior of the traders, in the context of FX markets.

Thus, the "Heterogeneous Multi Agents Model", based on the hypothesis of heterogenous markets, is a successful microstructure model describing big, liquid FX markets. However, since our goal was to construct the model as realistic as possible, it became rather complex. It seems that some of the less relevant parameters, like some trading restrictions, might be eliminated and thus the model be simplified without losing the prediction of stylized facts.

A Results of the Stress Testing

In this chapter we list the complete results of the stress testing discussed in Chapter 5.3.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
default set	3.45	3.96	4.40	4.53	5.03	6.14	[4.26, 4.80]
01 credit limit	3.70	4.22	4.55	4.64	4.96	6.43	[4.43, 4.84]
5 credit limit	3.76	4.24	4.88	4.81	5.20	6.46	[4.57, 5.04]
buy order	3.59	4.43	4.69	4.69	4.99	5.80	[4.49, 4.89]
sell order	3.63	4.23	4.54	4.67	5.23	5.95	[4.43, 4.91]
15 fund value	3.93	4.34	4.58	4.71	4.97	6.48	[4.50, 4.91]
075 fund value	3.85	4.45	4.84	4.94	5.28	6.48	[4.67, 5.21]
5 strategies	3.58	4.38	4.84	4.74	5.11	6.01	[4.54, 4.94]
20 strategies	4.08	4.36	4.62	4.67	4.90	5.37	[4.52, 4.82]
0 price spread	3.82	4.25	4.66	4.68	5.08	5.56	[4.50, 4.85]
1 price spread	3.62	4.09	4.47	4.57	5.03	5.87	[4.34, 4.80]
334 price weights	3.69	4.26	4.57	4.77	5.11	7.06	[4.49, 5.05]
343 price weights	3.68	4.20	4.52	4.66	5.15	6.30	[4.44, 4.88]
424 price weights	4.05	4.35	4.51	4.68	4.81	6.12	[4.49, 4.88]
43525 price weights	3.69	4.32	4.74	5.04	5.64	7.18	[4.71, 5.36]
523 price weights	3.58	3.94	4.30	4.33	4.73	6.26	[4.13, 4.53]
52525 price weights	3.66	4.14	4.68	4.73	5.25	6.27	[4.49, 4.97]
1000 traders	3.53	3.90	4.13	4.33	4.64	6.11	[4.10, 4.56]
2000 traders	3.36	4.74	5.05	5.24	5.61	8.48	[4.86, 5.62]
1 wealth weight	3.52	4.23	4.50	4.63	4.97	6.88	[4.38, 4.88]
3 wealth weight	3.89	4.27	4.56	4.57	4.85	5.34	[4.43, 4.72]

Table A.1: Estimation of α for all modified default sets.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
default set	2.40	2.85	3.11	3.08	3.31	3.59	[2.96, 3.20]
01 credit limit	2.32	2.84	3.09	3.08	3.21	3.80	[2.95, 3.21]
5 credit limit	2.37	2.71	2.94	2.96	3.27	3.72	[2.82, 3.09]
buy order	2.46	2.75	2.92	2.98	3.13	3.73	[2.87, 3.09]
sell order	2.41	2.70	2.95	3.01	3.15	3.76	[2.87, 3.14]
15 fund value	2.20	2.44	2.64	2.63	2.72	3.56	[2.52, 2.73]
075 fund value	2.77	3.13	3.39	3.46	3.69	4.87	[3.29, 3.63]
5 strategies	2.31	2.71	2.93	2.97	3.21	3.71	[2.85, 3.10]
20 strategies	2.40	2.65	2.89	2.87	3.02	3.46	[2.77, 2.97]
0 price spread	2.21	2.53	2.70	2.81	2.95	4.36	[2.64, 2.98]
1 price spread	2.64	3.16	3.37	3.36	3.63	4.26	[3.23, 3.49]
334 price weights	2.62	2.91	3.12	3.16	3.36	3.76	[3.05, 3.27]
343 price weights	2.37	2.73	3.06	3.02	3.24	3.68	[2.88, 3.16]
424 price weights	2.55	2.99	3.20	3.23	3.38	4.82	[3.07, 3.40]
43525 price weights	2.23	2.68	2.88	2.87	3.10	3.38	[2.76, 2.97]
523 price weights	2.62	2.91	3.12	3.16	3.36	3.76	[3.05, 3.27]
52525 price weights	2.23	2.65	2.92	2.94	3.16	3.80	[2.79, 3.08]
1000 traders	2.40	2.70	2.81	2.86	3.03	3.43	[2.75, 2.96]
2000 traders	2.62	2.92	3.09	3.19	3.43	4.01	[3.04, 3.33]
1 wealth weight	2.34	2.77	2.99	3.02	3.20	3.78	[2.90, 3.15]
3 wealth weight	2.41	2.68	2.84	2.88	3.08	3.52	[2.77, 2.99]

Table A.2: Estimation of n for all modified default sets.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
default set	0.271	0.294	0.313	0.319	0.340	0.399	[0.307, 0.332]
01 credit limit	0.258	0.304	0.315	0.320	0.341	0.408	[0.307, 0.333]
5 credit limit	0.264	0.298	0.331	0.332	0.357	0.405	[0.318, 0.347]
buy order	0.262	0.310	0.332	0.328	0.352	0.391	[0.318, 0.339]
sell order	0.261	0.308	0.330	0.327	0.358	0.396	[0.313, 0.340]
15 fund value	0.276	0.355	0.365	0.370	0.394	0.431	[0.358, 0.383]
075 fund value	0.203	0.265	0.287	0.286	0.310	0.348	[0.274, 0.299]
5 strategies	0.264	0.305	0.330	0.330	0.356	0.414	[0.317, 0.342]
20 strategies	0.282	0.321	0.335	0.340	0.364	0.399	[0.329, 0.352]
0 price spread	0.227	0.329	0.358	0.351	0.381	0.432	[0.334, 0.369]
1 price spread	0.232	0.271	0.289	0.294	0.309	0.363	[0.282, 0.305]
334 price weights	0.239	0.304	0.323	0.323	0.348	0.371	[0.312, 0.335]
343 price weights	0.267	0.301	0.319	0.327	0.355	0.408	[0.312, 0.341]
424 price weights	0.204	0.289	0.304	0.305	0.324	0.375	[0.292, 0.318]
43525 price weights	0.289	0.314	0.339	0.342	0.360	0.427	[0.330, 0.355]
523 price weights	0.260	0.290	0.311	0.309	0.333	0.366	[0.299, 0.319]
52525 price weights	0.258	0.307	0.333	0.335	0.365	0.425	[0.320, 0.351]
1000 traders	0.285	0.323	0.345	0.344	0.359	0.402	[0.332, 0.355]
2000 traders	0.245	0.284	0.313	0.309	0.331	0.365	[0.296, 0.321]
1 wealth weight	0.259	0.304	0.325	0.325	0.349	0.408	[0.312, 0.337]
3 wealth weight	0.277	0.314	0.341	0.340	0.361	0.398	[0.327, 0.352]

Table A.3: Estimation of h for all modified default sets.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
default set	0.517	0.539	0.550	0.554	0.569	0.588	[0.546, 0.561]
01 credit limit	0.509	0.543	0.556	0.557	0.571	0.582	[0.550, 0.563]
5 credit limit	0.504	0.540	0.549	0.550	0.565	0.583	[0.544, 0.557]
buy order	0.515	0.533	0.554	0.551	0.562	0.603	[0.542, 0.559]
sell order	0.505	0.540	0.548	0.549	0.564	0.578	[0.543, 0.555]
15 fund value	0.518	0.543	0.552	0.554	0.567	0.598	[0.547, 0.561]
075 fund value	0.514	0.542	0.555	0.555	0.567	0.589	[0.548, 0.562]
5 strategies	0.518	0.540	0.554	0.552	0.561	0.594	[0.545, 0.558]
20 strategies	0.506	0.543	0.556	0.553	0.564	0.593	[0.546, 0.560]
0 price spread	0.515	0.544	0.555	0.555	0.565	0.595	[0.548, 0.562]
1 price spread	0.490	0.541	0.555	0.553	0.573	0.592	[0.545, 0.562]
334 price weights	0.525	0.549	0.555	0.557	0.567	0.585	[0.551, 0.563]
343 price weights	0.506	0.528	0.542	0.545	0.563	0.591	[0.536, 0.553]
424 price weights	0.536	0.572	0.584	0.583	0.593	0.623	[0.575, 0.590]
43525 price weights	0.492	0.533	0.540	0.541	0.550	0.573	[0.534, 0.547]
523 price weights	0.541	0.565	0.579	0.578	0.591	0.610	[0.572, 0.584]
52525 price weights	0.508	0.553	0.569	0.565	0.574	0.605	[0.558, 0.571]
1000 traders	0.510	0.538	0.549	0.550	0.563	0.603	[0.543, 0.557]
2000 traders	0.493	0.533	0.551	0.549	0.565	0.587	[0.540, 0.557]
1 wealth weight	0.522	0.546	0.553	0.555	0.568	0.585	[0.549, 0.562]
3 wealth weight	0.506	0.550	0.561	0.557	0.567	0.606	[0.550, 0.565]

Table A.4: Estimation of D_1 for all modified default sets.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
default set	0.519	0.534	0.549	0.552	0.568	0.592	[0.544, 0.559]
01 credit limit	0.512	0.546	0.558	0.557	0.567	0.583	[0.550, 0.562]
5 credit limit	0.510	0.536	0.547	0.547	0.561	0.574	[0.541, 0.553]
buy order	0.508	0.532	0.548	0.547	0.560	0.570	[0.540, 0.555]
sell order	0.501	0.538	0.545	0.547	0.561	0.580	[0.541, 0.553]
15 fund value	0.518	0.539	0.550	0.551	0.564	0.590	[0.545, 0.558]
075 fund value	0.515	0.542	0.557	0.554	0.565	0.587	[0.547, 0.560]
5 strategies	0.511	0.539	0.554	0.550	0.559	0.579	[0.543, 0.556]
20 strategies	0.505	0.539	0.557	0.552	0.561	0.589	[0.544, 0.559]
0 price spread	0.515	0.538	0.555	0.553	0.566	0.589	[0.546, 0.560]
1 price spread	0.500	0.537	0.550	0.550	0.568	0.583	[0.543, 0.558]
334 price weights	0.516	0.541	0.555	0.554	0.565	0.583	[0.548, 0.560]
343 price weights	0.505	0.527	0.544	0.544	0.558	0.587	[0.536, 0.552]
424 price weights	0.538	0.564	0.580	0.577	0.593	0.610	[0.570, 0.583]
43525 price weights	0.508	0.529	0.539	0.540	0.549	0.580	[0.534, 0.546]
523 price weights	0.540	0.564	0.575	0.573	0.585	0.600	[0.568, 0.579]
52525 price weights	0.502	0.551	0.564	0.562	0.575	0.600	[0.556, 0.569]
1000 traders	0.514	0.541	0.551	0.550	0.557	0.597	[0.544, 0.556]
2000 traders	0.490	0.530	0.553	0.547	0.561	0.583	[0.539, 0.555]
1 wealth weight	0.523	0.542	0.555	0.554	0.565	0.578	[0.548, 0.559]
3 wealth weight	0.503	0.543	0.561	0.555	0.567	0.593	[0.548, 0.562]

Table A.5: Estimation of D_2 for all modified default sets.

	min	1st qu	median	mean	3rd qu	max	95% conf.int.
default set	0.511	0.530	0.548	0.549	0.565	0.598	[0.541, 0.558]
01 credit limit	0.513	0.544	0.554	0.553	0.562	0.583	[0.547, 0.560]
5 credit limit	0.512	0.532	0.545	0.543	0.556	0.576	[0.537, 0.549]
buy order	0.498	0.531	0.540	0.544	0.560	0.601	[0.536, 0.552]
sell order	0.497	0.534	0.543	0.545	0.558	0.581	[0.539, 0.551]
15 fund value	0.513	0.534	0.546	0.548	0.564	0.585	[0.541, 0.556]
075 fund value	0.517	0.539	0.555	0.552	0.563	0.584	[0.546, 0.558]
5 strategies	0.504	0.536	0.550	0.547	0.557	0.581	[0.540, 0.553]
20 strategies	0.501	0.536	0.553	0.550	0.561	0.592	[0.543, 0.558]
0 price spread	0.512	0.530	0.551	0.549	0.564	0.584	[0.542, 0.556]
1 price spread	0.505	0.534	0.547	0.546	0.565	0.575	[0.539, 0.554]
334 price weights	0.505	0.541	0.550	0.550	0.562	0.579	[0.543, 0.557]
343 price weights	0.499	0.523	0.544	0.542	0.558	0.589	[0.534, 0.550]
424 price weights	0.537	0.553	0.572	0.570	0.584	0.601	[0.563, 0.577]
43525 price weights	0.509	0.526	0.535	0.539	0.550	0.587	[0.533, 0.545]
523 price weights	0.543	0.556	0.569	0.568	0.578	0.591	[0.563, 0.573]
52525 price weights	0.498	0.547	0.561	0.560	0.579	0.596	[0.552, 0.567]
1000 traders	0.518	0.538	0.548	0.548	0.556	0.588	[0.542, 0.554]
2000 traders	0.486	0.525	0.551	0.544	0.558	0.581	[0.536, 0.552]
1 wealth weight	0.520	0.537	0.556	0.551	0.560	0.574	[0.545, 0.557]
3 wealth weight	0.500	0.541	0.560	0.552	0.568	0.580	[0.545, 0.560]

Table A.6: Estimation of D_3 for all modified default sets.

B S-PLUS Script

We list the "S-Plus" script used to run the "Heterogeneous Multi Agents Model" and to analyze the generated time series. The idea is that readers familiar with "S-Plus" may follow our thoughts. Where necessary, the script contains detailed comments.

```
# Run the HETEROGENEOUS MULTI AGENTS MODEL program and analyze the
# generated time series.

##### PARAMETERS OF THE HETEROGENEOUS MULTI AGENTS MODEL PROGRAM #########

# The next 5 parameters determine the various time scales and when we can
# speak of a weak/strong price in/decrease of the coded price process.
# They are mutually dependent and may, for a given TimeScaleHorizon, be
# calculated using the diffusion law discussed in the main text.

# Choose one of the following time scale horizon sets:

# time scale horizon set 1
#TimeScaleHorizon <- c( 1, 2, 4, 28, 120, 1440)
#IncrementPlus2   <- c( 0.0100, 0.0141, 0.0200, 0.0529, 0.1095, 0.3795)
#IncrementPlus1   <- c( 0.0025, 0.0035, 0.0050, 0.0132, 0.0274, 0.0949)
#IncrementMinus1  <- c(-0.0025,-0.0035,-0.0050,-0.0132,-0.0274,-0.0949)
#IncrementMinus2  <- c(-0.0100,-0.0141,-0.0200,-0.0529,-0.1095,-0.3795)

# time scale horizon set 2 (Produces bad results!)
#TimeScaleHorizon <- c( 1, 2, 4, 8, 16, 32)
#IncrementPlus2   <- c( 0.0100, 0.0141, 0.0200, 0.0283, 0.0400, 0.0566)
#IncrementPlus1   <- c( 0.0025, 0.0035, 0.0050, 0.0071, 0.0100, 0.0141)
#IncrementMinus1  <- c(-0.0100,-0.0141,-0.0200,-0.0283,-0.0400,-0.0566)
#IncrementMinus2  <- c(-0.0025,-0.0035,-0.0050,-0.0071,-0.0100,-0.0141)

# time scale horizon set 3 (default)
TimeScaleHorizon <- c( 1, 4, 16, 64, 256, 1024)
IncrementPlus2   <- c( 0.0100, 0.0200, 0.0400, 0.0800, 0.1600, 0.3200)
IncrementPlus1   <- c( 0.0025, 0.0050, 0.0100, 0.0200, 0.0400, 0.0800)
IncrementMinus1  <- c(-0.0025,-0.0050,-0.0100,-0.0200,-0.0400,-0.0800)
IncrementMinus2  <- c(-0.0100,-0.0200,-0.0400,-0.0800,-0.1600,-0.3200)

# time scale horizon set 4
#TimeScaleHorizon <- c( 1, 5, 25, 125, 625, 3125)
#IncrementPlus2   <- c( 0.0100, 0.0224, 0.0500, 0.1118, 0.2500, 0.5590)
```

```
#IncrementPlus1   <- c( 0.0025, 0.0056, 0.0125, 0.0280, 0.0625, 0.1398)
#IncrementMinus1  <- c(-0.0025,-0.0056,-0.0125,-0.0280,-0.0625,-0.1398)
#IncrementMinus2  <- c(-0.0100,-0.0224,-0.0500,-0.1118,-0.2500,-0.5590)

# The following parameters should in each column sum up to 1.

# Various price weights for time scales 1,2,...,6
PriceWeight1 <- c(0.4, 0.4, 0.4, 0.4, 0.4, 0.4)  # (default: 0.4)
PriceWeight2 <- c(0.3, 0.3, 0.3, 0.3, 0.3, 0.3)  # (default: 0.3)
PriceWeight3 <- c(0.3, 0.3, 0.3, 0.3, 0.3, 0.3)  # (default: 0.3)

# Lenght of the simulation and the corresponding price process
NbTradingRounds <- 30000  # (default: 100000)
PriceVector <- rep(0,NbTradingRounds)

# This set of parameters characterizes the FX-Model in detail (see also the
# C++ source code):
Parameters <- rep(0,17)
Parameters[1]  <- 1500   # NbTraders            (default: 1500)
Parameters[2]  <- Parameters[1] # NbOrders      (fixed: NbTraders)
Parameters[3]  <- 10     # NbStrategies         (>3, default: 10)
Parameters[4]  <- 125    # NbPriceProcessPaths  (fixed: 125)
Parameters[5]  <- Parameters[3]+1 # NbAccounts  (fixed: NbStrategies+1)
Parameters[6]  <- 6      # NbTimeScales         (fixed: 6)
Parameters[7]  <- 1.0    # InitFundValue        (default: 1.0)
Parameters[8]  <- 1.0    # InitMeanPrice        (default: 1.0)
Parameters[9]  <- 1      # DefaultOrderType     (default: 1)
Parameters[10] <- 10     # DefaultNbOfferedAssets (default: 10)
Parameters[11] <- 0.0002 # PriceSpreadHalf      (default: 0.0002)
Parameters[12] <- 0.0005 # PriceSpreadStdDev    (default: 0.0005)
Parameters[13] <- 0.001  # FundValueStdDev      (default: 0.001)
Parameters[14] <- 0.2    # WealthTradingWeight  (default: 0.2)
Parameters[15] <- 0.05   # CreditLimit          (default: 0.05)
Parameters[16] <- 10030  # RandomSeed           (default: 0=system time)
Parameters[17] <- 4      # Emas/TimeScale        (fixed: 4)

### START OF SIMULATION ##################################################

# Define the SPlus-function runFXmarket.
runFXmarket <- function()
{
    .C("runFXmarket",
    as.double(Parameters),
```

```
    as.integer(TimeScaleHorizon),
    as.double(IncrementPlus2),
    as.double(IncrementPlus1),
    as.double(IncrementMinus1),
    as.double(IncrementMinus2),
    as.double(PriceVector),
    as.integer(NbTradingRounds),
    as.double(PriceWeight1),
    as.double(PriceWeight2),
    as.double(PriceWeight3))[[7]] # return 7th argument
}

# Load the dll file of the FX-Model at the indicated path.
dll.load("d:\\workspace\\FXmarket.dll", "runFXmarket")

# Start the simulation and write the time series in a file.
write(runFXmarket(), "d:\\workspace\\pricevector", ncolumns=1)

# Unload the dll file.
dll.unload ("d:\\workspace\\FXmarket.dll", T)

### PARAMETERS FOR STATISTICAL TESTS ####################################

timeHorizon <- 500  # (default: 500)
histogramBars <- 50 # (default: 50)
highFreqInt <- 1    # (default: 1)
maxTime <- 1500     # (default:1000; smaller than nbTradingRounds)
q95   <- (NbTradingRounds - 95/100 * NbTradingRounds)

### STATISTICAL TESTS FOR THE VARIOUS STYLIZED FACTS #####################

# Read the price time series from the file "pricevector".
priceProcess <- scan("d:\\workspace\\pricevector")

### Price process ########################################################

plot(priceProcess, type='l')

### General description of time series ###################################

# Calculate logreturns
logReturns <- diff(log(priceProcess), highFreqInt)
```

```
### Asymmetry in information flow ########################################

fine.temp <- matrix(abs(logReturns), nrow=8)
fine <- colMeans(fine.temp)
fine.length<-length(fine)
coarse.temp <- matrix(logReturns, nrow=8)
coarse <- abs(colSums(coarse.temp))
coarse.length<-length(coarse)

for(u in 1:50)
{
fine.trunc1 <- fine[-c(1:(u+0))]
coarse.trunc1 <- coarse[-c((coarse.length +1-u):coarse.length)]
cor1 <- cor(fine.trunc1,coarse.trunc1)

fine.trunc2 <- fine[-c((fine.length +1-u):fine.length)]
coarse.trunc2 <- coarse[-c(1:(u+0))]
cor2 <- cor(fine.trunc2,coarse.trunc2)

print(cor1-cor2)
}

x <- coarse
y <- fine
mat <- cbind(x,y)
z <- acf(mat,20)

plot(0,0,xlim=c(-20,20), ylim=c(-0.1,0.6),
    xlab='Lag', ylab='10030', type='n')
lines(0:20, z$acf[,1,2], type='l')
lines(0:-20, z$acf[,2,1], type='l')
lines(0:20, z$acf[,1,2] - z$acf[,2,1], type='l', pch=3)
lines(c(-0,20),c(-0.0237, -0.0237), lty=2)
lines(c(-0,20),c(0.0235, 0.0235), lty=2)
abline(h=0, lty=1)
abline(v=0, lty=1)

### Volatility clustering ################################################

plot(logReturns, type ='l')

### Long memory ##########################################################
```

```
f <-acf(logReturns, timeHorizon-1)
g <- f[[1]][-1]
t <- seq(1,timeHorizon-1, by=1)

plot(g, type="h", xlim=c(0,timeHorizon), ylim=c(-0.1,0.2),
    xlab='Lag', ylab='ACF')
ACFconf.int95 <- 1.96/sqrt(length(logReturns))
lines(c(0, timeHorizon),c(-ACFconf.int95, -ACFconf.int95), lty=2)
lines(c(0, timeHorizon),c(ACFconf.int95, ACFconf.int95), lty=2)
abline(h=0, lty=1)


af <-acf(abs(logReturns), timeHorizon-1)
ag <- af[[1]][-1]
at <- seq(1,timeHorizon-1, by=1)

# For a long memory the confidence interval of b should NOT contain 0.
generalMemory <- nls(ag~m*(1+n*at)^(-1/n), start=list(m=0.5,n=1))
summary(generalMemory)
plot(residuals(generalMemory))

# estimate hyperbolic parameters
longMemoryHyper <- nls(ag~k*at^(-h), start=list(k=0.5,h=0.5))
summary(longMemoryHyper)
plot(residuals(longMemoryHyper))

plot(ag, type="l", xlim=c(0,timeHorizon), ylim=c(-0.05,0.25),
    xlab='Lag', ylab='ACF')
ACFconf.int95 <- 1.96/sqrt(length(logReturns))
lines(c(0, timeHorizon),c(-ACFconf.int95, -ACFconf.int95), lty=2)
lines(c(0, timeHorizon),c(ACFconf.int95, ACFconf.int95), lty=2)
abline(h=0, lty=1)

### Heavy Tails (Distribution) ###########################################

library(evis6)     # load package of McNeil
#emplot(logReturns,alog="xy")
#hist(logReturns, histogramBars)
plot(density(logReturns), type="l", main="Density of Logreturns", log="y")
qqnorm(logReturns)
qqline(logReturns)

# Estimation of the shape parameter using the POT model of McNeil
```

```
#shape(logReturns, models=30, start=15, end=q90, ci=0.95, table=T)
q95.tail  <- gpd(data=logReturns, nextremes=q95)
q95.tail

### Test for a scaling law (p=1) #########################################

# Create an empty vector
meanAbsLogReturns <- rep(0,maxTime)

# Calculate absolute log returns and their means over different time
# intervals.
for (i in 1:maxTime)
{
    meanAbsLogReturns[i] <- mean(abs(diff(log(priceProcess), i)))
}

# If the plot is a straight line, we have a scaling law.
plot(meanAbsLogReturns, type='l', log="xy")

# Create a vector with components denoting increasing time intervals
deltaT <- seq(1,maxTime, by=1)

# We fit the scaling law E[|logReturns|] = meanAbsLogReturns(deltaT) =
# c(deltaT)^D to the model log(E[|logReturns|]) = log(c(deltaT)^D) =
# log(c) + D*log(deltaT) + epsilon with gls and lm.
scalingExponentLm <- lm(log(meanAbsLogReturns) ~ log(deltaT))

# We find under "Coefficients -> log(deltaT)" an estimate for the drift
# exponent D and under "Coefficients -> (Intercept)" an estimate for
# log(c).
summary(scalingExponentLm)

### Test for a scaling law (p=2) #########################################

# Create an empty vector
rootedMeanSquaredAbsLogReturns<- rep(0,maxTime)

# Calculate absolute log returns and their means over different time
# intervals.
for (i in 1:maxTime)
{
    rootedMeanSquaredAbsLogReturns[i] <-
        (mean((diff(log(priceProcess), i))^2))^(1/2)
```

```
}

# If the plot is a straight line, we have a scaling law.
plot(rootedMeanSquaredAbsLogReturns, type='l', log="xy", xlab="delta-t",
    ylab="(E[|r(delta-t)|^2])^(1/2)")

# Create a vector with components denoting increasing time intervals
deltaT <- seq(1,maxTime, by=1)

# We fit the scaling law (E[|logReturns|^2])^(1/2) = c(deltaT)^D to the model
# log((E[|logReturns|^2])^(1/2)) = log(c) + D*log(deltaT) + epsilon
# with gls and lm.
scalingExponentLm2  <-
    lm(log(rootedMeanSquaredAbsLogReturns) ~ log(deltaT))

# We find under "Coefficients -> log(deltaT)" an estimate for the drift
# exponent D and under "Coefficients -> (Intercept)" an estimate for
# log(c).
summary(scalingExponentLm2)

### Test for a scaling law (p=3) ##########################################

# Create an empty vector
rootedMeanCubedAbsLogReturns<- rep(0,maxTime)

# Calculate absolute log returns and their means over different time
# intervals.
for (i in 1:maxTime)
{
    rootedMeanCubedAbsLogReturns[i] <-
        (mean((diff(log(priceProcess), i))^3))^(1/3)
}

# If the plot is a straight line, we have a scaling law.
plot(rootedMeanSquaredAbsLogReturns, type='l', log="xy", xlab="delta-t",
    ylab="(E[|r(delta-t)|^3])^(1/3)")

# Create a vector with components denoting increasing time intervals
deltaT <- seq(1,maxTime, by=1)

# We fit the scaling law E[|logReturns|^2] = c(deltaT)^D to the model
# log((E[|logReturns|^3])^(1/3)) = log(c) + D*log(deltaT) + epsilon
# with gls and lm.
```

```
scalingExponentLm2  <-
    lm(log(rootedMeanSquaredAbsLogReturns) ~ log(deltaT))

# We find under "Coefficients -> log(deltaT)" an estimate for the drift
# exponent d and under "Coefficients -> (Intercept)" an estimate for
# log(c).
summary(scalingExponentLm2)
```

C C++ Source Code of the Heterogeneous Multi Agents Model

The source code of the "Heterogeneous Multi Agents Model" is published under the General Public License

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

 HETEROGENEOUS MULTI AGENTS MODEL

 This program simulates the FX rate in a market with heterogeneous
 traders. The goal is to catch the stylized facts of FX markets.

 Copyright (C) 2002 by

 CHRISTOPH M. SCHMID,
 Institute for Mathematical Statistics, University of Bern,
 Sidlerstrasse 5, CH-3012 Bern
 or christoph.schmid@stat.unibe.ch

 and

 WOLFGANG BREYMANN,
 Risklab, Department of Mathematics, ETH-Zentrum, CH-8092 Zuerich
 or breymann@math.ethz.ch

 This program is free software; you can redistribute it and/or modify it
 under the terms of the GNU General Public License as published by the
 Free Software Foundation; either version 2 of the License, or (at your
 opinion) any later version.

 This program is distributed in the hope that it will be useful, but
 WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
 Public License for more details.

 You should have received a copy of the GNU General Public License along
 with this program; if not, write to the Free Software Foundation, Inc.,
 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

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

In the following we list the full source code, so that any reader familiar with C++ can reproduce the results presented in this thesis.

C.1 CodedPrice.h & CodedPrice.cpp

```
// CodedPrice.h -- header file for the class CodedPrice

#ifndef _CodedPrice_h_
#define _CodedPrice_h_

#include <iostream>
#include <vector>
#include "Mmc.h"
#include "Fifo.h"
#include "Ema.h"

using namespace std;

/*------------------------------------------------------------------------

CLASS

    CodedPrice

OVERVIEW TEXT

    This class manages the coding of the simulated price process. For a
    Given a time period, we decide whether the price has increased or
    decreased in a weak or strong way or whether it remained more or less
    constant. This is necessary in order to introduce technical trading
    strategies in the class Portfolio.

------------------------------------------------------------------------*/

class CodedPrice {

public:

/// GROUP: Constructor and destructor
```

```
    CodedPrice(const Fifo& fifo);

    ~CodedPrice();

/// GROUP: Get and set

    // Calculate the latest consulted prices.
    void MeanPrices(int timeScale);

    void GetMeanPrices (
        double* v,
        int timeScale) const;

    double GetLatestMeanPrice(int timeScale) const
        { return latestMeanPrice_[timeScale]; }

    double GetSecondLatestMeanPrice(int timeScale) const
        { return secondLatestMeanPrice_[timeScale]; }

    double GetThirdLatestMeanPrice(int timeScale) const
        { return thirdLatestMeanPrice_[timeScale]; }

    double GetCodedPriceProcess(int timeScale) const
        { return codedPriceProcess_[timeScale]; }

    // Calculate the coded price increments.
    void PriceIncrements(int timeScale);

    // Every sequence of coded price increments determines a new coded
    // price process.
    void NewCodedPriceProcess(int timeScale);

    // Predict the price using ema.
    void EmaPriceTrend(int timeScale);

    double GetEmaPriceTrend(int timeScale);

/// GROUP: Update

    // Update emas in each trading round.
    void EmaUpdate(int timeScale);

    // Recode the price process on the different time scales.
```

```
    void updatePriceHistory();

private:

    // encoding the price
    int encodeReturn(
        double r,
        int timeScale);

    // Reference to an object of type Fifo, that saves the latest
    // 3*MMC::scalePriceCode_[timeScale].timeScaleHorizon_ + 2 mean market
    // prices. The fifo is updated only in the class Mamm.
    const Fifo& fifo_;

    // Saving of the latest consulted prices of the various time scales
    // using emas (index 0 = time scale with shortest horizon).
    // Those change every trading round and are necessary to determine the
    // coded price process.
    vector<double> emaMeanPrice_;
    vector<double> latestMeanPrice_;
    vector<double> secondLatestMeanPrice_;
    vector<double> thirdLatestMeanPrice_;

    // Saving of the coded price increment over the latest time periods.
    // Since this is again time scale dependent, we use vector<>.
    vector<int> codedPriceIncrement_;
    vector<int> latestCodedPriceIncrement_;
    vector<int> secondLatestCodedPriceIncrement_;

    // Saving of the (time scale dependent) coded price process,.
    vector<double> codedPriceProcess_;

    // To smooth the price process, we use 6*4=24 emas (for all time scales and
    // every consulted past price, incl. present meanPrice) while coding the
    // price process.
    vector< vector<Ema*> > emas_;

    // Prediction of the price using time scale dependent emas.
    vector<Ema*> emaPriceTrend_;

};

#endif _CodedPrice_h_
```

```
// CodedPrice.cpp -- implementation file for class CodedPrice

#include "CodedPrice.h"

/// GROUP: Constructor and destructor

CodedPrice::CodedPrice(
    const Fifo& fifo)
  : fifo_(fifo),
    emaMeanPrice_(MMC::nbTimeScales_, MMC::initMeanPrice_),
    latestMeanPrice_(MMC::nbTimeScales_, MMC::initMeanPrice_),
    secondLatestMeanPrice_(MMC::nbTimeScales_, MMC::initMeanPrice_),
    thirdLatestMeanPrice_(MMC::nbTimeScales_, MMC::initMeanPrice_),
    codedPriceIncrement_(MMC::nbTimeScales_, 0),
    latestCodedPriceIncrement_(MMC::nbTimeScales_, 0),
    secondLatestCodedPriceIncrement_(MMC::nbTimeScales_, 0),
    codedPriceProcess_(MMC::nbTimeScales_, 62),
    emas_(MMC::nbTimeScales_),
    emaPriceTrend_(MMC::nbTimeScales_)
{
    // Set correct size of emas_ and initialization.
    for (int timeScale = 0; timeScale < MMC::nbTimeScales_; timeScale++) {
        emas_[timeScale].resize(MMC::nbEmas_);
        for (int emaIndex = 0; emaIndex < MMC::nbEmas_; emaIndex++) {
            emas_[timeScale][emaIndex] = new Ema(MMC::initMeanPrice_,
                MMC::scalePriceCode_[timeScale].timeScaleHorizon_);
        }
        emaPriceTrend_.resize(MMC::nbTimeScales_);
        emaPriceTrend_[timeScale] = new Ema(MMC::initMeanPrice_,
            MMC::scalePriceCode_[timeScale].timeScaleHorizon_);
    }
}

CodedPrice::~CodedPrice()
{
    for (int timeScale = 0; timeScale < MMC::nbTimeScales_; timeScale++) {
        for (int emaIndex = 0; emaIndex < MMC::nbEmas_; emaIndex++) {
            delete emas_[timeScale][emaIndex];
        }
        delete emaPriceTrend_[timeScale];
    }
}
```

```
/// GROUP: Get and set

void CodedPrice::MeanPrices(int timeScale)
{
    emaMeanPrice_[timeScale] = emas_[timeScale][0]->value();
    latestMeanPrice_[timeScale] = emas_[timeScale][1]->value();
    secondLatestMeanPrice_[timeScale] = emas_[timeScale][2]->value();
    thirdLatestMeanPrice_[timeScale] = emas_[timeScale][3]->value();
}

void CodedPrice::GetMeanPrices(
    double* v,
    int timeScale) const
{
    v[0] = latestMeanPrice_[timeScale];
    v[1] = secondLatestMeanPrice_[timeScale];
    v[2] = thirdLatestMeanPrice_[timeScale];
}

void CodedPrice::PriceIncrements(int timeScale)
{
    // Local variable relIncr1 stores relative price increment.
    double relIncr1 = (emaMeanPrice_[timeScale]
        - latestMeanPrice_[timeScale])
            / emaMeanPrice_[timeScale];

    codedPriceIncrement_[timeScale] =
        encodeReturn(relIncr1, timeScale);

    // Local variable relIncr2 stores relative price increment.
    double relIncr2 = ( latestMeanPrice_[timeScale]
        - secondLatestMeanPrice_[timeScale] )
            / latestMeanPrice_[timeScale];

    latestCodedPriceIncrement_[timeScale] =
        encodeReturn(relIncr2, timeScale);

    // Local variable relIncr3 stores relative price increment.
    double relIncr3 = ( secondLatestMeanPrice_[timeScale]
        - thirdLatestMeanPrice_[timeScale] )
            / secondLatestMeanPrice_[timeScale];
```

```
    secondLatestCodedPriceIncrement_[timeScale] =
        encodeReturn(relIncr3, timeScale);
}

void CodedPrice::NewCodedPriceProcess(int timeScale)
{
    // The chosen weights (the earlier the increment the less weight) are
    // reasonable as long as the corresponding technical strategies are
    // chosen in a pure random way. I.e. if it does not matter whether
    // to a coded price history of e.g. -2/-2/-2 <=> 0 a buy, hold or sell
    // is recommended.
    // The algorithm yields: 0/25/50/75/100 + 0/5/10/15/20 + 0/1/2/3/4
    // = 0/1/2.../123/124.
    codedPriceProcess_[timeScale] =
        (codedPriceIncrement_[timeScale] + 2) * 25
        + (latestCodedPriceIncrement_[timeScale] + 2) * 5
        + (secondLatestCodedPriceIncrement_[timeScale] + 2);
}

void CodedPrice::EmaPriceTrend(int timeScale)
{
    emaPriceTrend_[timeScale]->step(fifo_[0]);
}

double CodedPrice::GetEmaPriceTrend(int timeScale)
{
    return emaPriceTrend_[timeScale]->value();
}

/// GROUP: Update

void CodedPrice::updatePriceHistory()
{
    for (int timeScale = 0; timeScale < MMC::nbTimeScales_; timeScale ++) {
        EmaUpdate(timeScale);
        EmaPriceTrend(timeScale);
        MeanPrices (timeScale);
        PriceIncrements(timeScale);
        NewCodedPriceProcess(timeScale);
    }
}
```

```
void CodedPrice::EmaUpdate(int timeScale)
{
    long i1 = 3 * MMC::maxTimeHorizon_;
    long i2;

    i2 = MMC::scalePriceCode_[timeScale].timeScaleHorizon_;
    for (int emaIndex = 0; emaIndex < MMC::nbEmas_; emaIndex++) {
        emas_[timeScale][emaIndex]->step(fifo_[1 + emaIndex * i2]);
    }
}

int CodedPrice::encodeReturn(
    double r,
    int timeScale)
{
    if (r >= MMC::scalePriceCode_[timeScale].incrementPlus2_) {
        return 2;
    } else if (r >= MMC::scalePriceCode_[timeScale].incrementPlus1_) {
        return 1;
    } else if (r > MMC::scalePriceCode_[timeScale].incrementMinus1_) {
        return 0;
    } else if (r > MMC::scalePriceCode_[timeScale].incrementMinus2_) {
        return -1;
    } else {
        return -2;
    }
}
```

C.2 Ema.h

```
// Ema.h -- header file for the class Ema

#ifndef _Ema_h_
#define _Ema_h_

#include <cmath>
#include "Mmc.h"

/*------------------------------------------------------------------------
```

```
CLASS

    Ema

DESCRIPTION

    The very core of the exponential moving average (EMA) algorithm, is
    the iteration

    ema[x](t+1) = \mu * ema[x](t)  + ( 1-\mu ) * x(t),

    where \mu = \exp{-1/\tau}.

    Note that a time step of 1 is assumed for the iteration.

------------------------------------------------------------------------*/

class Ema {

public:

    friend class CodedPrice;

/// GROUP: Constructor

    Ema(
        const double initVal,
        const double tau)
  : ema_(initVal)
{
    mu(tau);
}

/// GROUP: Methods

    // Set exponential factor.
    void mu(const double tau)
        { mu_ = exp(-1.0/tau); }

    // Ema iteration.
    double step(double x)
        { ema_ = mu_*ema_ + (1-mu_)*x; return ema_; }
```

```
    // Get value of ema.
    const double value() const
        { return ema_; }

private:

    double ema_;
    double mu_;

};

#endif _Ema_h_
```

C.3 Fifo.h & Fifo.cpp

```
// Fifo.h -- header file for the class Fifo

#ifndef _Fifo_h_
#define _Fifo_h_

#include <vector>

using namespace std;

/*--------------------------------------------------------------------------

CLASS

    Fifo

OVERVIEW TEXT

    This class creates first-in-first-out objects of type double.
    I.e. the first number that enters the fifo leaves it as first, too.

--------------------------------------------------------------------------*/

class Fifo {

public:
```

```
/// GROUP: Constructor

    Fifo (
        int size,
        double initValue);

/// GROUP: Methods

    // Write new value x in fifo.
    void operator()(double x);

    // Access fifo at n-th position; n=0 returns the newest value, n=1 the
    // second newest and so on.
    double operator[](int n) const;

private:

    vector <double> queue_;
    int size_;
    int stackPointer_;
    double initValue_;

};

#endif _Fifo_h_


// Fifo.cpp -- implementation file for the class Fifo

#include <iostream>
#include "Fifo.h"

/// GROUP: Constructor

Fifo::Fifo (int size, double initValue)
  : size_(size),
    stackPointer_(0),
    queue_(size, initValue),
    initValue_(initValue)
{
    if ( size < 1 ) {
    cerr << "Error in Fifo:\n";
```

```
    cerr << "Size must be positive.\n";
    exit(0);
    }
}

/// GROUP: Methods

void Fifo::operator()(double x)
{
    stackPointer_--;
    if (stackPointer_ < 0) {    // computes modulo
        stackPointer_ += size_;
    }
    queue_[stackPointer_] = x;
}

double Fifo::operator[](int n) const
{
    if (n >= size_) {cerr << "n to large\n";}
    int i = stackPointer_ + n;
    if ( i  >= size_) {         // computes modulo
        i -= size_;
    }
    return queue_[i];
}
```

C.4 GaussRng.h & GaussRng.cpp

```
// GaussRng.h -- Produces Gaussian pseudo-random numbers.

#ifndef _GaussRng_h_
#define _GaussRng_h_

#include "UniformRng.h"

/*--------------------------------------------------------------------------

CLASS

    UniformRng
```

```
OVERVIEW TEXT

    This from the base class RngBasis derived class produces Gaussian
    random numbers. The underlying algorithm is marsaglia.

--------------------------------------------------------------------------*/

class GaussRng : public RngBasis {

public:

/// GROUP: Constructor

    // Default
    GaussRng();

    // Initialization by passing a seed.
    GaussRng(unsigned long seed);

    virtual ~GaussRng(){};

/// GROUP: Methods

    // Draw a random number.
    virtual double rand();

private:

    UniformRng uniformRng_;
    bool    nextOK_;
    double next_;

    // Internal state has to be prepared before generating random numbers.
    virtual void init();

};

#endif _GaussRng_h_


// GaussRng.cpp -- implementation file for the class GaussRng
```

```
#include <cmath>
#include "GaussRng.h"

/// GROUP: Constructor

GaussRng::GaussRng()
  : uniformRng_(),
    nextOK_(false),
    next_(0.0)
{
    init( );
}

GaussRng::GaussRng(unsigned long seed)
  : uniformRng_(seed),
    nextOK_(false),
    next_(0.0)
{
    init( );
}

/// GROUP: Methods

void GaussRng::init()
{
    next_ = 0;
    nextOK_ = false;
}

double GaussRng::rand()
{
    double v1, v2, fac, rsq;
    if (nextOK_) {
        nextOK_ = false;
        return next_;
    }
    do {
        v1 = 2.0 * uniformRng_.rand() - 1.0;   // -1 <= v1 <= 1
        v2 = 2.0 * uniformRng_.rand() - 1.0;   // -1 <= v2 <= 1
        rsq = v1 * v1 + v2 * v2;
    } while (rsq >= 1.0 || rsq <= 0.0);

    fac = sqrt(-2.0 * log(rsq) / rsq);
```

```
    next_ = v1 * fac;
    fac   = v2 * fac;
    nextOK_ = true;
    return fac;
}
```

C.5 Information.h

```
// Information.h -- header file for the class Information.

#ifndef _Information_h_
#define _Information_h_

#include <iostream>
#include <cmath>
#include "Mmc.h"
#include "GaussRng.h"
#include "Rng.h"

using namespace std;

/*-----------------------------------------------------------------------

CLASS

    Information

OVERVIEW TEXT

    The class Information simulates the fundamental value of the asset.
    Logreturns of the value follow a Gauss law.

-----------------------------------------------------------------------*/

class Information {

public:

/// GROUP: Constructor
```

```
    Information(
        double epsilon,
        double pastFundValue,
        double fundValue)
  : epsilon_(epsilon),
    pastFundValue_(pastFundValue),
    fundValue_(fundValue),
    pRng_(Rng::create())
{}

/// GROUP: Methods (inlined for efficiency reasons)

    // Update the new fundamental value at the begin of a trading round.
    void update() {
        LatestFundValue();              // information gets old
        Epsilon(pRng_->gaussianRand()); // random news
        FundValue(); }                  // new fundamental value

    void Epsilon(double gaussianRandomNumber)
        { epsilon_ = (gaussianRandomNumber * MMC::fundValueStdDev_); }

    void LatestFundValue()
        { pastFundValue_ = fundValue_; }

    void FundValue()
    { fundValue_ = exp(epsilon_) * pastFundValue_; }

    double GetLatestFundValue() const
        { return pastFundValue_; }

    double GetFundValue() const
        { return fundValue_; }

private:

    double epsilon_;
    double pastFundValue_;
    double fundValue_;
    Rng* pRng_;

};
```

```
#endif _Information_h_
```

C.6 Mmc.h & Mmc.cpp

```
// Mmc.h -- header file for the class MMC

#ifndef _MMC_h_
#define _MMC_h_

#include <stdlib.h>
#include <vector>
#include "UniformRng.h"
#include "GaussRng.h"

using namespace std;

// Some global statements

typedef enum  {buy, hold, sell} ordertype;

/*------------------------------------------------------------------------

CLASS

    MMC

OVERVIEW TEXT

    The class MMC (for market model constants) contains all parameters of
    the HETEROGENEOUS FX MARKET PROGRAM.

------------------------------------------------------------------------*/

class MMC {

public:

    // The following structure determines the constants for coding the
    // price process. They are time scale dependent.
    struct scaleConstants
```

```
{
    long timeScaleHorizon_;
    double incrementPlus2_;
    double incrementPlus1_;
    double incrementMinus1_;
    double incrementMinus2_;
};

// Setting of the model parameters.
static void set(
    double* pParameters,
    int* pTimeScaleHorizon,
    double* pIncrementPlus2,
    double* pIncrementPlus1,
    double* pIncrementMinus1,
    double* pIncrementMinus2,
    double* pPriceWeight1,
    double* pPriceWeight2,
    double* pPriceWeight3);

// Time horizon of traders with the longest time scale horizon.
static long maxTimeHorizon_;

// Number of traders in the market.
static long nbTraders_;

// Each trader enters one order, is thus equal to nbTraders_.
static long nbOrders_;

// Number of strategies stored in a portfolio of strategies.
static long nbStrategies_;

// Is given by the brain size of the traders (e.g 3) and the number
// of different price increments (e.g 5) (= 5^3).
static long nbPossiblePriceProcessPaths_;

// Number of virtual accounts plus 1 real account of a trader.
static long nbAccounts_;

// Index of real account.
static long iRWealth_;

// Number of time scales on which the traders act.
```

```
static long nbTimeScales_;

// Initial fundamental value..
static double initFundValue_;

// Initial mean price.
static double initMeanPrice_;

// Default type of order.
static ordertype defaultOrderType_;

// Default number of offered assets.
static long defaultNbOfferedAssets_;

// Weight coefficients of price offer components; is scale dependent.
static vector<double> priceWeight1_;
static vector<double> priceWeight2_;
static vector<double> priceWeight3_;

// Half expected price spread.
static double priceSpreadHalf_;

// Standard deviation of price spread.
static double priceSpreadStdDev_;

// Standard deviation of the logarithmic fundamental price changes.
static double fundValueStdDev_;

// The number of offered assets depends on the real wealth.
static double wealthTradingWeight_;

// If the security margin is 20%, then creditLimit_ = 0.2.
static double creditLimit_;

// Set random seed.
static long rndSeed_;

// Number of emas acting on a time scale (incl. present price ).
static long nbEmas_;

// A structure of scale constants for each time scale.
static vector<scaleConstants> scalePriceCode_;
```

```
    // The price is measured in basis points.
    static double Truncate_(double x);

};

#endif _MMC_h_
```

```
// Mmc.cpp -- implementation file of the class MMC

#include "Mmc.h"

// to be initialized in a set function
long MMC::maxTimeHorizon_;
long MMC::nbTraders_;
long MMC::nbOrders_;
long MMC::nbStrategies_;
long MMC::nbPossiblePriceProcessPaths_;
long MMC::nbAccounts_;
long MMC::iRWealth_;
long MMC::nbTimeScales_;

double MMC::initFundValue_;
double MMC::initMeanPrice_;

ordertype MMC::defaultOrderType_;
long MMC::defaultNbOfferedAssets_;

vector<double> MMC::priceWeight1_;
vector<double> MMC::priceWeight2_;
vector<double> MMC::priceWeight3_;

double MMC::priceSpreadHalf_;
double MMC::priceSpreadStdDev_;

double MMC::fundValueStdDev_;

double MMC::wealthTradingWeight_;

double MMC::creditLimit_;

long MMC::rndSeed_;
```

```
long MMC::nbEmas_;

vector<MMC::scaleConstants> MMC::scalePriceCode_;

// Use in run.cpp.
void MMC::set(
    double* pParameters,
    int* pTimeScaleHorizon,
    double* pIncrementPlus2,
    double* pIncrementPlus1,
    double* pIncrementMinus1,
    double* pIncrementMinus2,
    double* pPriceWeight1,
    double* pPriceWeight2,
    double* pPriceWeight3)
{
    nbTraders_ = long(pParameters[0]);
    nbOrders_ = long(pParameters[1]);
    nbStrategies_ = long(pParameters[2]);
    nbPossiblePriceProcessPaths_ = long(pParameters[3]);
    nbAccounts_ = long(pParameters[4]);
    iRWealth_ = 0;
    nbTimeScales_ = long(pParameters[5]);
    initFundValue_ = pParameters[6];
    initMeanPrice_ = pParameters[7];
    defaultOrderType_ = ordertype( int( pParameters[8] ) );
    defaultNbOfferedAssets_ = long(pParameters[9]);
    priceSpreadHalf_ = pParameters[10];
    priceSpreadStdDev_ = pParameters[11];
    fundValueStdDev_ = pParameters[12];
    wealthTradingWeight_= pParameters[13];
    creditLimit_ = pParameters[14];
    rndSeed_ = pParameters[15];
    nbEmas_ = pParameters[16];

    scalePriceCode_.resize(nbTimeScales_);
    for (int i = 0; i < nbTimeScales_; i++) {
        scalePriceCode_[i].timeScaleHorizon_ = pTimeScaleHorizon[i];
        scalePriceCode_[i].incrementPlus2_ = pIncrementPlus2[i];
        scalePriceCode_[i].incrementPlus1_ = pIncrementPlus1[i];
        scalePriceCode_[i].incrementMinus1_ = pIncrementMinus1[i];
        scalePriceCode_[i].incrementMinus2_ = pIncrementMinus2[i];
```

```
    }

    priceWeight1_.resize(nbTimeScales_);
    priceWeight2_.resize(nbTimeScales_);
    priceWeight3_.resize(nbTimeScales_);
    for (int j = 0; j < nbTimeScales_; j++) {
        priceWeight1_[j] = pPriceWeight1[j];
        priceWeight2_[j] = pPriceWeight2[j];
        priceWeight3_[j] = pPriceWeight3[j];
    }
}

double MMC::Truncate_(double x)
{
    return double(int(x*10000))/10000;
}
```

C.7 Mamm.h & Mamm.cpp

```
// Mamm.h -- header file for the class Mamm

#ifndef _Mamm_h_
#define _Mamm_h_

#include "Information.h"
#include "OrderBook.h"
#include "Portfolio.h"
#include "Trader.h"
#include "CodedPrice.h"
#include "Fifo.h"

using namespace std;

/*-----------------------------------------------------------------------

CLASS

    Mamm

OVERVIEW TEXT
```

```
    Main class that manages the multi-agent market model.

--------------------------------------------------------------------------*/

class Mamm {

public:

/// Group: Destructor

    ~Mamm();

/// GROUP: Methods

    static Mamm* create(
        double* pParameters,
        int* pTimeScaleHorizon,
        double* pIncrementPlus2,
        double* pIncrementPlus1,
        double* pIncrementMinus1,
        double* pIncrementMinus2,
        double* pPriceWeight1,
        double* pPriceWeight2,
        double* pPriceWeight3);

    // Returns price
    double nextRound();

private:

    /// Private because first values have to be attribued to MMC.
    Mamm();

    // The order of the members is important because of initialization!
    Fifo fifo_;

    Information information_;
    CodedPrice priceProcess_; // created with default constructor

    OrderBook orderBook_;     // created with default constructor
    Trader* traders_;         // array of traders
    int traderId_;            // individualize the traders
```

```
    long tradingRound_;       // counter

};

#endif _Mamm_h_


// Mamm.cpp -- implementation file for the class Mamm

#include "Mmc.h"
#include "Mamm.h"

Mamm::Mamm()
  : fifo_(3*MMC::maxTimeHorizon_ + 2, MMC::initMeanPrice_),
    information_(
        MMC::initFundValue_,
        MMC::initFundValue_,
       MMC::initFundValue_),
    priceProcess_(fifo_),  // cf. comment in CodedPrice.h
    orderBook_(information_, priceProcess_),
    traderId_(0),
    tradingRound_(0)
{
    // Create an array of traders_.
    traders_ = new Trader[MMC::nbTraders_]();
    orderBook_.SetPtTrader(&traders_[0]);

    // Individualize the traders_.
    for (traderId_ = 0; traderId_ < MMC::nbTraders_; traderId_++) {
        traders_[traderId_].InitializeTrader(
            &information_,
            &orderBook_,
            &priceProcess_,
            traderId_);
    }
}

Mamm::~Mamm()
{
    delete[] traders_;
}

Mamm* Mamm::create(
```

```
    double* pParameters,
    int* pTimeScaleHorizon,
    double* pIncrementPlus2,
    double* pIncrementPlus1,
    double* pIncrementMinus1,
    double* pIncrementMinus2,
    double* pPriceWeight1,
    double* pPriceWeight2,
    double* pPriceWeight3)
{
    // Setting of all parameters of the model.
    MMC::set(pParameters,
            pTimeScaleHorizon,
            pIncrementPlus2,
            pIncrementPlus1,
            pIncrementMinus1,
            pIncrementMinus2,
            pPriceWeight1,
            pPriceWeight2,
            pPriceWeight3);

    MMC::maxTimeHorizon_
        = MMC::scalePriceCode_[MMC::nbTimeScales_ - 1].timeScaleHorizon_;

    // Create object;
    return new Mamm;
}

double Mamm::nextRound()
{
    tradingRound_++;
    orderBook_.initRound();
    information_.update();
    priceProcess_.updatePriceHistory();

    // Begin: Contact ALL traders_ in a randomly determined way.
    //        Check whether they are not bankrupt and if they are active,
    //        enter an order try to deal:
    orderBook_.ShuffleContactOrder();

    for (int contactedOrder = 0; // Loop over traders_
        contactedOrder < MMC::nbTraders_;
        contactedOrder++)
```

```
    {
        traderId_ = orderBook_.GetContactedTrader(contactedOrder);
        // Check if trader is NOT bankrupt.
        if ( !traders_[traderId_].bankruptcyRisk() )
        {
            // Check if trader is active.
            if ( traders_[traderId_].isActive(tradingRound_) )
            {
                // REAL STRATEGY:
                // Choose active strategy and trade
                traders_[traderId_].selectStrategy();
                traders_[traderId_].makeOrder(
                    traders_[traderId_].GetTraderTimeScale(),
                    information_.GetFundValue(),
                    priceProcess_.GetEmaPriceTrend(
                        traders_[traderId_].GetTraderTimeScale())
                );

                // VIRTUAL STRATEGIES:
                // We now trade with all virtual accounts. There is NO
                // INTERACTION with the ORDER BOOK but the number of
                // assets and the amount of money in the virtual account
                // do change. We note that VIRTUAL STRATEGIES ARE ALWAYS
                // EXECUTED IN FULL.
                traders_[traderId_].updateVirtualAccounts();
            }
        }
        else // If the trader is bankrupt, initialize a new trader with
             // the same Id:
        {
            traders_[traderId_].ReplaceTrader(
                &information_,
                &priceProcess_,
                traderId_);
        }
    }
    orderBook_.MeanPrice();
    fifo_(orderBook_.GetMeanPrice());

    return orderBook_.GetMeanPrice();
}
```

C.8 OrderBook.h & OrderBook.cpp

```
// OrderBook.h -- header file for the class Order_book

#ifndef _OrderBook_h_
#define _OrderBook_h_

#include <iostream>
#include <vector>
#include "Mmc.h"
#include "Information.h"
#include "CodedPrice.h"
#include "Rng.h"

using namespace std;

class Trader;
class Portfolio;

/*-------------------------------------------------------------------------

CLASS

    OrderBook

OVERVIEW TEXT

    The class OrderBook defines the struct order, which determines of
    whether to buy or sell, the number of assets to be traded and at which
    price. This class further manages the clearing of the orders.
    This procedure has to be done for the real account of a trader as well
    as for all his virtual accounts. This basket of accounts is introduced
    in the class Portfolio.

-------------------------------------------------------------------------*/

class OrderBook {

public:

    friend class Trader;
```

```
/// GROUP: Constructor and destructor

    OrderBook(
        const Information& info,
        const CodedPrice& process);

    ~OrderBook();


/// GROUP: Set and initialize

    void initRound();

    void AddedPricesAndTradeCounter();

    void MeanPrice();

    // Pseudopermutation of contact order.
    void ShuffleContactOrder();

    void SetPtTrader(Trader* pTrader)
        { pTrader_ = pTrader; }

    void VirtualOrderType(
        int traderId,
        int accountIndex,
        int codedPriceProcess);

    void NbOfferedAssets(
        int traderId,
        int orderBookIndex);

/// GROUP: Accessors

    int GetTradeCounter() const
        { return tradeCounter_; }

    double GetMeanPrice() const
        { return meanPrice_; }

    int GetContactedTrader( int contactedOrder ) const
        { return contactOrder_[contactedOrder]; }
```

```
    double GetPrice() const
        { return price_; }

    double GetPriceOffer( int orderBookIndex ) const
        { return book_[orderBookIndex].priceOffer_; }

    int GetOrderType( int orderBookIndex ) const
        { return book_[orderBookIndex].orderType_; }

/// GROUP: Computations

    // Rounded price offer.
    void PriceOffer(
        int orderBookIndex,
        int timeScale,
        double fundValue,
        double emaPriceTrend);

    // Reset the number of offered assets if a trading restriction is
    // violated.
    void TradingRestrictions(
        int nbAssets,
        int orderBookIndex,
        double money);

    // New entry in the order book.
    void OrderBook::NewEntry(
        int traderId,
        ordertype oType,
        int nbAssets,
        double price,
        int timeScale,
        double fundValue,
        double emaPriceTrend);

private:

    // Internally used types.
    // A single order consists of the trader's id-number, the type of
    // order (buy or sell), the offered price and the number of demanded
    // or offered assets.
    struct order
    {
```

```
        int traderId_;
        ordertype orderType_;
        double priceOffer_;
        int assets_;
    };

    // Helper function
    template <class T>
    void Trade (T x, T& a, T& b) { a  += x; b -= x; }

    // Attempt to clear the new order.
    void Clearing(
        int orderBookIndex,
        int traderId,
        int accountIndex);

    // References and pointers to collaborator objects.
    Rng* pRng_;      // global random number generators.
    const Information& information_;
    const CodedPrice& priceProcess_;
    Trader* pTrader_; // pointer to a specific trader

    // Variables
    int tradeCounter_;   // Counts number of trades.
    double addedPrices_; // Aggregated prices within 1 trading round.
    double meanPrice_;   // Mean of prices over 1 trading round.
    double price_;       // If a trade occurs we get a new price.

    // Necessary for trading with the virtual accounts.
    ordertype virtualOrderType_;
    double virtualPriceOffer_;
    double virtualNbOfferedAssets_;

    // The order book is an array of MMC::nbTraders_ structures of type
    // order. In each trading round, which means contacting all traders,
    // every trader enters his order.
    // (The sequence of contacting is randomly determined in each round.)
    // After each entry we immediately try to clear the new order. If we
    // are successful, we get a NEW PRICE. If we fail, the order remains
    // maximal 1 further round in the book (until it gets overwritten by
    // another order of, usually, another trader).
    vector<order> book_;
```

```
    // In each trading round the traders are contacted in a randomly
    // determined sequence.
    vector<int> contactOrder_;

};

#endif _OrderBook_h_



// OrderBook.cpp -- implementation file for the class Order_book


#include "OrderBook.h"
#include "CodedPrice.h"
#include "Trader.h"

/// GROUP: Constructor and destructor

OrderBook::OrderBook(
    const Information& info,
    const CodedPrice& process)
  : pRng_(Rng::create()),
    information_(info),
    priceProcess_(process),
    tradeCounter_ (0),
    addedPrices_(0),
    meanPrice_ (MMC::initMeanPrice_),
    price_ (MMC::initMeanPrice_),
    virtualOrderType_ (MMC::defaultOrderType_),
    virtualPriceOffer_ (0),
    virtualNbOfferedAssets_(0),
    book_(MMC::nbTraders_),
    contactOrder_(MMC::nbTraders_)
{
    for(int i = 0; i < MMC::nbTraders_; i++) {
        book_[i].orderType_ = MMC::defaultOrderType_;
        book_[i].priceOffer_ = MMC::initMeanPrice_;
        book_[i].assets_ = MMC::defaultNbOfferedAssets_;
    }
    for(int j = 0; j < MMC::nbTraders_; j++) {
        contactOrder_[j] = j;
    }
```

```
}

OrderBook::~OrderBook()
{
    pRng_->destroy();
}

/// GROUP: Set and initialize

void OrderBook::initRound()
{
    tradeCounter_ = 0;
    addedPrices_ = 0;
}

void OrderBook::AddedPricesAndTradeCounter()
{
    addedPrices_ += price_;
    tradeCounter_ ++;
}

void OrderBook::MeanPrice()
{
    if (tradeCounter_ > 0) {
        meanPrice_ = addedPrices_ / tradeCounter_;
        meanPrice_ = MMC::Truncate_(meanPrice_);
    }
}

void OrderBook::ShuffleContactOrder()
{
    double uniformRnd;
    for (int i = 0; i < MMC::nbTraders_; i++) {
        uniformRnd = pRng_->uniformRand();
        int change1 = (uniformRnd * MMC::nbTraders_);

        uniformRnd = pRng_->uniformRand();
        int change2 = (uniformRnd * MMC::nbTraders_);

        int auxiliary1 = contactOrder_[change1];
        int auxiliary2 = contactOrder_[change2];

        contactOrder_[change1] = auxiliary2;
```

```
        contactOrder_[change2] = auxiliary1;
    }
}

/// GROUP: Computations

void OrderBook::PriceOffer(
    int orderBookIndex,
    int timeScale,
    double fundValue,
    double emaPriceTrend)
{
   // Local variable priceOffer.
    double priceOffer = (book_[orderBookIndex].orderType_ == buy) ?
        (MMC::priceWeight1_[timeScale] * meanPrice_
        + MMC::priceWeight2_[timeScale] * fundValue
        + MMC::priceWeight3_[timeScale] * emaPriceTrend)
        * exp(- (pRng_->gaussianRand() * MMC::priceSpreadStdDev_
        + MMC::priceSpreadHalf_))
    :
        (MMC::priceWeight1_[timeScale] * meanPrice_
        + MMC::priceWeight2_[timeScale] * fundValue
        + MMC::priceWeight3_[timeScale] * emaPriceTrend)
        * exp(+ (pRng_->gaussianRand() * MMC::priceSpreadStdDev_
        + MMC::priceSpreadHalf_));

    book_[orderBookIndex].priceOffer_ = priceOffer;
    book_[orderBookIndex].priceOffer_ = MMC::Truncate_(
        book_[orderBookIndex].priceOffer_);
}

void OrderBook::NbOfferedAssets(
    int traderId,
    int orderBookIndex)
{
    book_[orderBookIndex].assets_ =
        MMC::wealthTradingWeight_ *
            pTrader_[traderId].GetPortfolio().AccountBasket(0).wealth_
            / meanPrice_;
}

void OrderBook::TradingRestrictions(
    int nbAssets,
```

```
    int orderBookIndex,
    double money)
{
    switch (book_[orderBookIndex].orderType_)
    {
        case buy:
            // Test whether the buy restriction does not hold,
            // i.e. whether a trader can not cover at least e.g. 5% of
            // the credit with his wealth.
            if (
                MMC::creditLimit_ *
                book_[orderBookIndex].assets_ *
                book_[orderBookIndex].priceOffer_
                    > nbAssets * meanPrice_ + money)
            {
                // Resize the number of offered assets.
                if (nbAssets * meanPrice_ + money > 0)
                {
                    book_[orderBookIndex].assets_ =
                        (nbAssets * meanPrice_ + money) /
                            (MMC::creditLimit_ *
                            book_[orderBookIndex].priceOffer_);
                } else {
                    book_[orderBookIndex].assets_=
                        -1 * (nbAssets * meanPrice_ + money) /
                            ( MMC::creditLimit_ *
                            book_[orderBookIndex].priceOffer_);
                }
            }
            break;

        case hold:
            book_[orderBookIndex].assets_ = 0;
            break;

        case sell:
            // Test whether the sell restriction does not hold,
            // i.e. whether a trader can not cover at least e.g 5% of
            // the short sell with his wealth.
            if (
                MMC::creditLimit_ *
                book_[orderBookIndex].assets_ * meanPrice_
                    > money + nbAssets * meanPrice_)
```

```
            {
                // Resize the number of offered assets:
                if (money + nbAssets * meanPrice_ > 0)
                {
                    book_[orderBookIndex].assets_ =
                        (money + nbAssets * meanPrice_) /
                            (MMC::creditLimit_ * meanPrice_);
                } else {
                    book_[orderBookIndex].assets_ =
                        -1 * (money + nbAssets * meanPrice_) /
                            (MMC::creditLimit_ * meanPrice_);
                }
            }
            break;
    }
}

void OrderBook::NewEntry(
    int traderId,
    ordertype oType,
    int nbAssets,
    double money,
    int timeScale,
    double fundValue,
    double emaPriceTrend)
{
    book_[traderId].orderType_ = oType; // traderId = orderBookIndex
    PriceOffer( traderId, timeScale, fundValue, emaPriceTrend );
    NbOfferedAssets(traderId, traderId);
    TradingRestrictions(nbAssets, traderId, money);

    // Is clearing possible?
    Clearing( traderId, traderId, MMC::iRWealth_);
    AddedPricesAndTradeCounter();
}

void OrderBook::Clearing(
    int bookId,
    int traderId,
    int accountIndex)
{
    // Compare new order(asking trader with index: bookId) with old
    // orders (answering trader with index: oldId):
```

```
    ordertype oType = book_[bookId].orderType_;
    double tradingPrice = book_[bookId].priceOffer_;
    Portfolio::account& accountNew = const_cast<Portfolio::account&>(
        pTrader_[traderId].GetPortfolio().AccountBasket(accountIndex)
    ); // Below we need to write to this object.

    for (int oldId = 0; oldId < MMC::nbTraders_; oldId++) {
        // Check whether the new order matches with an old one, i.e. if
        // they have equal priceOffer_ but different orderType_(except
        // from hold!).
        if (
            book_[bookId].priceOffer_ == book_[oldId].priceOffer_
            && book_[bookId].orderType_ != hold
            && book_[oldId].orderType_ != hold
            && book_[bookId].orderType_ != book_[oldId].orderType_)
        {
            Portfolio::account& accountOld =
                const_cast<Portfolio::account&>(
                    pTrader_[oldId].GetPortfolio().AccountBasket(
                        accountIndex));
            int tradedAssets;
            book_[bookId].assets_ < book_[oldId].assets_ ?
                tradedAssets = book_[bookId].assets_
                : tradedAssets = book_[oldId].assets_;
            double volume = tradedAssets * tradingPrice;

//-----------------------------------------------------------------------

switch (oType)
{
    case buy :
        // New entry asks for less or equal number of assets than old
        // order offers: Modify the portfolio of both traders:
        Trade(tradedAssets, accountNew.nbAssets_, accountOld.nbAssets_);
        Trade(volume, accountOld.money_, accountNew.money_);
        break;
    case sell :
        // New entry offers less or equal assets than old order asks for
        Trade(tradedAssets, accountOld.nbAssets_, accountNew.nbAssets_);
        Trade(volume, accountNew.money_, accountOld.money_);
        break;
}
```

```
//------------------------------------------------------------------------

        if ( book_[bookId].assets_ <= book_[oldId].assets_ )
        {   // New entry offers less assets than old order:
            // Old order only partially executed
            book_[oldId].assets_ -= book_[bookId].assets_;
            oType = hold; // trading successfully completed
        }
        else
        {//The symmetric case: new order only partially executed.
            book_[bookId].assets_ -= book_[oldId].assets_;
            book_[oldId].orderType_ = hold;
        }
        price_ = book_[bookId].priceOffer_;
        }
    }
}
```

C.9 Portfolio.h & Portfolio.cpp

```
// Portfolio.h -- header file for the class Portfolio

#ifndef _Portfolio_h_
#define _Portfolio_h_

#include <iostream>
#include <vector>
#include "Mmc.h"
#include "GaussRng.h"
#include "Rng.h"

using namespace std;

/*--------------------------------------------------------------------------

CLASS

    Portfolio
```

```
OVERVIEW TEXT

    Handles the real and virtual accounts as well as trading strategies.

-------------------------------------------------------------------------*/

class OrderBook;

class Portfolio {

public:

    // An account consists of money, an integer number of assets and the
    // corresponding wealth, depending on the market price of the asset.
    struct account
    {
        double money_;
        int nbAssets_;
        double wealth_;
    };

/// GROUP: Constructors and destructor

    Portfolio();

    Portfolio(double price);

    ~Portfolio();

/// GROUP: Methods

    void InitializeFundStrategy(
        double meanPrice,
        double fundValue);

    void InitializeTechnicalStrategies();

    // The suggestion of whether to buy, hold or sell is depending on the
    //strategy and the coded price history.
    ordertype GetOrderType(
        int strategyIndex,
        int priceCode) const
    { return ordertype (strategyBasket_[strategyIndex][priceCode]); }
```

```
    ordertype GetOrderType(int codedPriceProcess) const
        { return ordertype (
            strategyBasket_[activeStrategy_][codedPriceProcess]); }

    double GetMoney(int accountIndex) const
        { return accountBasket_[accountIndex].money_; }

    double GetNbAssets(int accountIndex) const
        { return accountBasket_[accountIndex].nbAssets_; }

    void Wealth(
        int accountIndex,
        double price)
    {
        accountBasket_[accountIndex].wealth_ =
            accountBasket_[accountIndex].money_
            + accountBasket_[accountIndex].nbAssets_ * price;
    }

    // Choose the strategy of the most successful virtual account.
    void selectStrategy(double price);

    void ActiveStrategy();

    int GetActiveStrategy() const
        { return activeStrategy_; }

    // Access a specific account of the account basket.
    const account& AccountBasket(int accountIndex) const
        { return accountBasket_[accountIndex]; }

    // Calculate amount of money and number of assets of virtual accounts.
    void VirtualAccountUpdate(
        int accountIndex,
        double priceOffer,
        long codedPriceProcess,
        int nbVirtualAssets);

private:

    // Pointer to the random number generators.
    // Has to be declared first because of order of initialization
```

```
    // in constructor
    Rng* pRng_;

/// Account part

    // Definition of an array of MMC::nbAccounts_ - 1 virtual accounts
    // and 1 real account with index iRWealth=0.
    vector<account> accountBasket_;

/// Strategy part

    // MMC::nbStrategies_ strategies.
    vector< vector <int> > strategyBasket_;

    // Index of strategy with highest virtual wealth (>0):
    int activeStrategy_;

};

#endif _Portfolio_h_
```

```
// Portfolio.cpp -- implementation file for the class Portfolio

#include "Portfolio.h"
#include "OrderBook.h"

/// GROUP: Constructors and destructor

Portfolio::Portfolio()
  : pRng_(Rng::create())
{}

Portfolio::Portfolio(double price)
  : pRng_(Rng::create()),
    accountBasket_(MMC::nbAccounts_),
    strategyBasket_(MMC::nbStrategies_),
    activeStrategy_(10 * pRng_->uniformRand())
{
    int i; // account index
    for (i = 0; i < MMC::nbAccounts_; i++){
        // We assume that the initial amount of money is Gaussian
```

```
        // distributed with an expectation value of 100 and a standard
        // deviation of 30. (We allow only positive amounts of money.)
        do {
            accountBasket_[i].money_ = pRng_->gaussianRand() * 30 + 100;
        } while (accountBasket_[i].money_ < 0);

        // We assume that the initial amount of money is Gaussian
        // distributed with an expectation value of 100 and a standard
        // deviation of 30. (We allow only positive amounts of money.)
        do {
            accountBasket_[i].nbAssets_ = pRng_->gaussianRand()* 30 + 100;
        } while (accountBasket_[i].nbAssets_ < 0);

        accountBasket_[i].wealth_ =
            accountBasket_[i].money_ + accountBasket_[i].nbAssets_ *price;
    }

    // Set correct size of strategyBasket_.
    for (i = 0; i < MMC::nbStrategies_; i++) {
        strategyBasket_[i].resize(MMC::nbPossiblePriceProcessPaths_);
    }
}

Portfolio::~Portfolio()
{
    pRng_->destroy();
}

/// GROUP: Methods

void Portfolio::InitializeFundStrategy(
    double meanPrice,
    double fundValue)
{
    int j;// coded price process path index

    // implementation of the fundamental strategy [0]
    for (j = 0; j < MMC::nbPossiblePriceProcessPaths_; j++) {
        if (meanPrice - fundValue >= 0.005) {
            strategyBasket_[0][j] = 2;
        } else if (meanPrice - fundValue <= -0.005) {
            strategyBasket_[0][j] = 0;
        } else {
```

```
            strategyBasket_[0][j] = 1;
        }
    }
}

void Portfolio::InitializeTechnicalStrategies()
{
    double uniformRnd;
    int i; // strategy index
    int j;  // price process path index

    // Implementation of the optimistic strategy [1]
    for (j = 0; j < MMC::nbPossiblePriceProcessPaths_; j++) {
        strategyBasket_[1][j] = buy;
    }
    // implementation of the pessimistic strategy [2]
    for (j = 0; j < MMC::nbPossiblePriceProcessPaths_; j++) {
        strategyBasket_[2][j] = sell;
    }

    // Implementation of (MMC::nbStrategies_ - 3) technical strategies
    for (i = 3; i < MMC::nbStrategies_; i++) {
        for (j = 0; j < MMC::nbPossiblePriceProcessPaths_; j++) {
            uniformRnd = pRng_->uniformRand();

            if (uniformRnd < 0.333333333333333) {
                strategyBasket_[i][j] = buy;
            } else if (uniformRnd >= 0.666666666666666) {
                strategyBasket_[i][j] = sell;
            } else {
                strategyBasket_[i][j] = hold;
            }
        }
    }
}

/// Strategy part

void Portfolio::selectStrategy(double price)
{
    for (
        int accountIndex = 1;
        accountIndex < MMC::nbAccounts_;
```

```
        accountIndex++)
    {
        Wealth ( accountIndex, price );
        ActiveStrategy();
    }
}

void Portfolio::ActiveStrategy()
{
    // Fundamental strategy (corresponding to virtual account with
    // accountIndex i=1) is the default active strategy.
    activeStrategy_ = 0;

    // Check whether a technical strategy is more successful.
    for (int i = 2; i < MMC::nbAccounts_; i++) {
        if (accountBasket_[i].wealth_ >
            accountBasket_[activeStrategy_].wealth_)
        {
            activeStrategy_ = i-1;
        }
    }
}

void Portfolio::VirtualAccountUpdate (
    int accountIndex,
    double priceOffer,
    long codedPriceProcess,
    int nbVirtualAssets)
{
    // Remember that strategy_index = accountIndex -1, see portfolio.cpp
    ordertype type =
        ordertype (strategyBasket_[accountIndex-1][codedPriceProcess]);
    if ( type == hold ) {
        return;
    }

    double r = priceOffer * nbVirtualAssets;
    accountBasket_[accountIndex].money_ += ( type == sell ) ? r : -r;
    accountBasket_[accountIndex].nbAssets_ += ( type == buy ) ?
    nbVirtualAssets
    : - nbVirtualAssets;
}
```

C.10 Rng.h & Rng.cpp

```
// Rng.h -- header file for the class Rng

#ifndef _Rng_h_
#define _Rng_h_

#include <iostream>
#include "UniformRng.h"
#include "GaussRng.h"
#include "Mmc.h"

using namespace std;


/*--------------------------------------------------------------------------

CLASS

    Rng

OVERVIEW TEXT

    Singleton class containing random number generators for class Trader.
    The goal is that only 1 Gauss and 1 uniform random number generator
    object exist; because they need a lot of resources.

--------------------------------------------------------------------------*/

class Rng {

public:

    static Rng* create();

    void destroy();

    // Draw next uniform random number.
    double uniformRand()
        { return pRndUni_->rand(); }

    // Draw next gaussian random number
```

```
    double gaussianRand()
        { return pRndGauss_->rand(); }

private:

    Rng();

    ~Rng();

    static int nbTraders_;
    static Rng* pRng_;

    UniformRng* pRndUni_;
    GaussRng* pRndGauss_;

};

#endif _Rng_h_
```

```
// Rng.cpp -- implementation file for the class Rng

#include "Rng.h"

int Rng::nbTraders_ = 0;
Rng* Rng::pRng_ = NULL;

Rng::Rng()// Set random seed
{
    if (MMC::rndSeed_ == 0) {
        pRndUni_ = new UniformRng(time(0));
        pRndGauss_ = new GaussRng(time(0));
    } else {
        pRndUni_ = new UniformRng(MMC::rndSeed_);
        pRndGauss_ = new GaussRng(MMC::rndSeed_);
    }
}

Rng* Rng::create()
{
    if (nbTraders_ == 0) {
        pRng_ = new Rng;
```

```
    }
    nbTraders_++;
    return pRng_;
}

void Rng::destroy()
{
    if (nbTraders_ != 1) {
        nbTraders_--;
    } else {
        delete pRng_;
    }
}

Rng::~Rng()
{
    delete pRndUni_;
    delete pRndGauss_;
}
```

C.11 RngBasis.h & RngBasis.cpp

```
// RngBasis.h -- header file for the class RngBasis

#ifndef _RngBasis_h_
#define _RngBasis_h_

#include <ctime>

/*------------------------------------------------------------------------

CLASS

    RngBasis

OVERVIEW TEXT

    RngBasis is a class to generate pseudo-random numbers.

    The classes inherited from this base class produce pseudo-random
```

```
    numbers of a specific distribution.

    There are methods for generating a seed from the system time
    (randomSeed), set and retrieve the seed explicitly and it also
    provides a trivial algorithm (Park Miller) that "randomizes" the
    system time in order to provide a useful initial random seed.
    Furthermore, this abstract base class provides the general interface
    to all random number generator classes.

------------------------------------------------------------------------*/


class RngBasis {

public:

/// GROUP: Constructors and destructor

    // This constructor generates a random seed.
    RngBasis();

    // Constructor with seed.
    RngBasis(unsigned long);

    virtual ~RngBasis();

/// GROUP: Methods

    // Return the current seed.
    virtual unsigned long seed() const
        {return seed_;};

    // Set a seed.
    virtual void setSeed(unsigned long);

    // Generate a random seed with the system time.
    unsigned long randomSeed();

protected:

    // A minimal algorithm to "randomize" the system time.
    unsigned long parkMiller(unsigned long seed);
```

```
    // Used by parkMiller.
    static const long  pmg_;

    // Initialize the internal states, if necessary.
    // Must be overloaded by the inherited class.
    virtual void init() {};

    // The base from where random numbers are generated.
    unsigned long seed_;

    // Some constants.
    static unsigned long MaxCard32_; // = 4294967295UL; // 2^32-1
    static double        m32Double_; // = ((double) MaxCard32);

};

#endif _RngBasis_h_


// RngBasis.cpp -- implementation file for the class RngBasis

#include <climits>
#include "RngBasis.h"

const long RngBasis::pmg_ = 16807;

RngBasis::RngBasis()
{
    setSeed(0);
}

RngBasis::RngBasis(unsigned long seed)
{
    setSeed(seed);
}

RngBasis::~RngBasis()
{
}

void RngBasis::setSeed(unsigned long seed)
{
    if (seed == 0) {
```

```
        seed_ = randomSeed();
    }
    else {
        seed_ = seed;
    }
    init();
}

unsigned long RngBasis::randomSeed()
{
    unsigned long seed;
    seed = time(0);
    return parkMiller(seed);
}

unsigned long RngBasis::parkMiller(unsigned long seed)
{
    if (seed <= (LONG_MAX / pmg_)) {
        seed = (seed * pmg_) % LONG_MAX;
    }
    else {
        long high_part = seed / (LONG_MAX / pmg_);
        long low_part  = seed % (LONG_MAX / pmg_);

        long test = pmg_ * low_part - (LONG_MAX % pmg_) * high_part;

        if (test > 0) {
            seed = test;
        }
        else {
            seed = test + LONG_MAX;
        }
    }
    return 2 * seed;
}

unsigned long RngBasis::MaxCard32_ = 4294967295UL; // 2^32-1
double RngBasis::m32Double_ = ((double) MaxCard32_);
```

C.12 RunFXdll.cpp

```
// RunFXdll.cpp -- simulates the price process of a FX market

#include <iostream>
#include "Mamm.h"

using namespace std;

extern "C"
{

int runFXmarket(
    double* pParameters,
    int* pTimeScaleHorizon,
    double* pIncrementPlus2,
    double* pIncrementPlus1,
    double* pIncrementMinus1,
    double* pIncrementMinus2,
    double* pPriceVector,
    int* pTradingRounds,
    double* pPriceWeight1,
    double* pPriceWeight2,
    double* pPriceWeight3,
    double* pPriceWeight4,
    double* pTrendWeight1,
    double* pTrendWeight2,
    double* pTrendWeight3)
{
    Mamm* pMarketModel = Mamm::create(
        pParameters,
        pTimeScaleHorizon,
        pIncrementPlus2,
        pIncrementPlus1,
        pIncrementMinus1,
        pIncrementMinus2,
        pPriceWeight1,
        pPriceWeight2,
        pPriceWeight3);

    // Start trading.
```

```
    for(
        int tradingRound = 0;
        tradingRound < *pTradingRounds;
        tradingRound ++)
    {
        pPriceVector[tradingRound] = pMarketModel->nextRound();
    }

    delete pMarketModel;

    return 0;
}

} // end extern "C"
```

C.13 RunFXconsole.h & RunFXconsole.cpp

```
// runFXconsole.h -- makes the program console compatible

extern "C"
int runFXmarket(
    double* pParameters,
    long* pTimeScaleHorizon,
    double* pIncrementPlus2,
    double* pIncrementPlus1,
    double* pIncrementMinus1,
    double* pIncrementMinus2,
    double* pPriceVector,
    long* pTradingRounds,
    double* pPriceWeight1,
    double* pPriceWeight2,
    double* pPriceWeight3);


// RunFXconsole.cpp -- This file and RunFXconsole.h make the program
//                     console compatible.

#include <iostream>
#include <fstream>
#include <time.h>
```

```
#include "RunFXconsole.h"

using namespace std;

// THE FOLLOWING PARAMETERS MAY BE ADJUSTED. NOTE SOME CONSTRAINTS!
static double pParameters[17] = {
1000   // NbTraders            (default: 1000)
,1000  // NbOrders             (fixed: NbTraders)
,10    // NbStrategies         (>3, default: 10)
,125   // NbPossiblePriceProcessPaths (fixed: 125)
,11    // NbAccounts           (fixed: NbPriceProcessPaths+1)
,6     // NbTimeScales         (fixed: 6)
,1.0   // InitFundValue        (default: 1.0)
,1.0   // InitMeanPrice        (default: 1.0)
,1     // DefaultOrderType     (default: 1)
,10    // DefaultNbOfferedAssets (default: 10)
,0.01  // PriceSpreadHalf      (default: 0.01)
,0.03  // PriceSpreadStdDev    (default: 0.03)
,0.001 // FundValueStdDev      (default: 0.001)
,0.20  // WealthTradingWeight  (default: 0.2)
,0.05  // CreditLimit          (default: 0.05)
,0     // RandomSeed           (default: 0=system time; else = seed)
,4     // Emas/TimeScale       (fixed: 4)
};

// The following lines characterize the time scales.
static long pTimeScaleHorizon[6] =
     {1, 2, 4, 28, 120, 1440};
static double pIncrementPlus2[6] =
     {0.010, 0.014, 0.020, 0.053, 0.110, 0.380};
static double pIncrementPlus1[6] =
     {0.0025, 0.0035, 0.0050, 0.0130, 0.0270, 0.0950};
static double pIncrementMinus1[6] =
     {-0.0025, -0.0035, -0.0050, -0.0130, -0.0270, -0.0950};
static double pIncrementMinus2[6] =
     {-0.010, -0.014, -0.020, -0.053, -0.110, -0.380};

static double pPriceWeight1[6] = {0.4, 0.4, 0.4, 0.4, 0.4, 0.4};
static double pPriceWeight2[6] = {0.3, 0.3, 0.3, 0.3, 0.3, 0.3};
static double pPriceWeight3[6] = {0.3, 0.3, 0.3, 0.3, 0.3, 0.3};

int main()
{
```

```
    char filename[20];
    cout << "Enter the name for a new file:\n";
    cin >> filename;

    cout << "Enter the number of trading rounds:\n";
    long pNbTradingRounds;
    cin >> pNbTradingRounds;

    double* pPriceVector = new double[pNbTradingRounds];

    clock_t t_begin = clock();
    runFXmarket(
        &pParameters[0],
        &pTimeScaleHorizon[0],
        &pIncrementPlus2[0],
        &pIncrementPlus1[0],
        &pIncrementMinus1[0],
        &pIncrementMinus2[0],
        pPriceVector,
        &pNbTradingRounds,
        &pPriceWeight1[0],
        &pPriceWeight2[0],
        &pPriceWeight3[0]);

    clock_t t_end = clock();

    ofstream fout(filename);
    int width();
    fout.width(10);
    for(int i = 0; i < pNbTradingRounds; i++) {
        fout << pPriceVector[i] << "\n";
    }

    fout.close();
    delete[] pPriceVector;
        cout << "CPU time of subroutine runFXmarket: "
            << (double(t_end) - t_begin)/CLOCKS_PER_SEC
            << " seconds." << endl;
        cout << "program finished." << endl;
    return 0;
}
```

C.14 Trader.h & Trader.cpp

```
// Trader.h -- header file for the class Trader

#ifndef _Trader_h_
#define _Trader_h_

#include <iostream>
#include "Mmc.h"
#include "Rng.h"
#include "Portfolio.h"
#include "Information.h"
#include "OrderBook.h"
#include "UniformRng.h"

using namespace std;

/*-------------------------------------------------------------------------

CLASS

    Trader

OVERVIEW TEXT

    This class characterizes a trader in detail.

-------------------------------------------------------------------------*/

class CodedPrice;
class Information;

class Trader {

public:

/// GROUP: Constructor

    Trader(){};

/// GROUP: Initialization
```

```
    void InitializeTrader(
        const Information* info,
        OrderBook* book,
        const CodedPrice* pPriceProcess,
        int traderId);

/// GROUP: Accessors

    int GetTraderTimeScale() const
        { return timeScaleIndex_; }

    const Portfolio& GetPortfolio()
        { return portfolio_; }

/// GROUP: Actions

    void selectStrategy()
        { portfolio_.selectStrategy( pOrderBook_->GetPrice() ); }

    void updateVirtualAccounts();

    void makeOrder(
        int timeScale,
        double fundValue,
        double emaPriceTrend);

    // Replacing a bankrupt trader (initialize a new trader except of the
    // trader_id): The new trader keeps the trader_id of the old one. But
    // he gets new technical strategies and virtual portfolios as well as
    // an initial amount of money and an initial number of assets.
    // Further he may act on a different time scale.
    void ReplaceTrader(
        Information* pInformation,
        CodedPrice* pPriceProcess,
        int traderId);

/// GROUP: Control

    bool isActive(int tradingRound);

    bool bankruptcyRisk();

private:
```

```
// Pointers and references to collaborator classes which does not
// belong to trader.
Rng* pRng_;

OrderBook* pOrderBook_;
const Information* pInformation_;
const CodedPrice* pPriceProcess_;

// This helper function initializes the timeScale_
// and the initialTraderActivityStatus_.
void initializeTimeScale(
    double uniRnd1,
    double uniRnd2);

// Identification number of trader.
int traderId_;

// The time scale determines how many trading rounds a trader waits
// until he gets acitive again.
// The time scales are uniformly distributed.
int timeScale_;

// Enumeration of timeScale_. The smaller the timeScale_ the smaller
// the index.
int timeScaleIndex_;

// Trader bankruptcy state is depending on the real wealth.
bool isTraderBankrupt_;

// The risk of bankruptcy (0 = no debts, 1 = for the 1st time a
// negative real wealth, 2 =.., e.g. 50 = bankruptcy).
int traderBankruptcyRisk_;

// Shows whether a trader tries to deal and is time scale dependent.
bool isTraderActive_;

// Randomly determined initial state of the activity of a trader.
int initTraderActivityStatus_;

// Every trader possesses a portfolio of strategies.
Portfolio portfolio_;
```

```
};

#endif _Trader_h_


// Trader.cpp -- implementation file of the class Trader

#include "Trader.h"
#include "OrderBook.h"
#include "Information.h"
#include "CodedPrice.h"

/// GROUP: Initialization

void Trader::InitializeTrader(
    const Information* pInformation,
    OrderBook* pOrderBook,
    const CodedPrice* pPriceProcess,
    int traderId)
{
    traderId_ = traderId;

    pRng_ = Rng::create();
    portfolio_ = Portfolio::Portfolio(MMC::initMeanPrice_);
    double uniRnd1 = pRng_->uniformRand();
    double uniRnd2 = pRng_->uniformRand();

    initializeTimeScale(uniRnd1, uniRnd2);
    traderBankruptcyRisk_ = 0;
    isTraderBankrupt_ = false;

    isTraderActive_ = (initTraderActivityStatus_ == 0) ? true : false;

    pOrderBook_ = pOrderBook;
    pInformation_ = pInformation;
    pPriceProcess_ = pPriceProcess;

    double meanPrice = pOrderBook_->GetPrice();
    portfolio_.InitializeFundStrategy(
        meanPrice,
        pInformation_->GetFundValue());
    portfolio_.InitializeTechnicalStrategies();
}
```

```
/// GROUP: Actions

void Trader::updateVirtualAccounts()
{
    double virtualPriceOffer = pOrderBook_->GetPriceOffer(traderId_);
    double codedPriceProcess = pPriceProcess_->GetCodedPriceProcess(
        timeScaleIndex_ );
    for (
        int accountIndex = 1;
        accountIndex < MMC::nbAccounts_;
        accountIndex++)
    {
        double virtualNbOfferedAssets =
            (MMC::wealthTradingWeight_ *
                portfolio_.AccountBasket(accountIndex).wealth_)
             / pOrderBook_->GetPrice();
        virtualNbOfferedAssets = int (virtualNbOfferedAssets);

        // Modify virtual accounts>
        portfolio_.VirtualAccountUpdate(
            accountIndex,
            virtualPriceOffer,
            codedPriceProcess,
            virtualNbOfferedAssets);
    }
}

void Trader::makeOrder(
    int timeScale,
    double fundValue,
    double emaPriceTrend)
{
    ordertype oType =
        portfolio_.GetOrderType(
            pPriceProcess_->GetCodedPriceProcess(timeScaleIndex_));
    int nbAssets = portfolio_.GetNbAssets(MMC::iRWealth_);
    double money = portfolio_.GetMoney(MMC::iRWealth_);

    pOrderBook_->NewEntry(
        traderId_,
        oType,
        nbAssets,
```

```
        money,
        timeScale,
        fundValue,
        emaPriceTrend);
}

void Trader::ReplaceTrader(
    Information* pInformation,
    CodedPrice* pPriceProcess,
    int traderId)
{
    portfolio_ = Portfolio::Portfolio(MMC::initMeanPrice_);
    double uniRnd1 = pRng_->uniformRand();
    double uniRnd2 = pRng_->uniformRand();

    InitializeTrader(
        pInformation,
        pOrderBook_,
        pPriceProcess,
        traderId);
}

/// GROUP: Control

bool Trader::isActive(int tradingRound)
{
    return (
        isTraderActive_ =
            (tradingRound % timeScale_ == initTraderActivityStatus_) ?
            true : false);
}

bool Trader::bankruptcyRisk()
{
    // Calculate real wealth (index 0) of trader.
    portfolio_.Wealth(0, pOrderBook_->GetPrice());

    // Calculate trader bankruptcy risk.
    if (portfolio_.AccountBasket(0).wealth_ < 0) {
        traderBankruptcyRisk_ += 1;
    } else {
        traderBankruptcyRisk_ = 0;
    }
```

```
    // Check whether trader is bankrupt.
    return (
        isTraderBankrupt_ = (traderBankruptcyRisk_ > 50) ?
            true : false);
}

void Trader::initializeTimeScale(
    double uniRnd1,
    double uniRnd2)
{
    timeScaleIndex_ = int(uniRnd1 * MMC::nbTimeScales_);
    timeScale_ = MMC::scalePriceCode_[timeScaleIndex_].timeScaleHorizon_;
    initTraderActivityStatus_ = int(uniRnd2 * timeScale_);
}
```

C.15 UniformRng.h & UniformRng.cpp

```
// UniformRng.h -- Produces uniformly distributed random numbers.
//                 R(471, 1568, 6988, 9689, Xor).


#ifndef _UniformRng_h_
#define _UniformRng_h_

#include "RngBasis.h"

/*-------------------------------------------------------------------------

CLASS

    UniformRng

OVERVIEW TEXT

    This from the base class RngBasis derived class produces uniform
    random numbers.

-------------------------------------------------------------------------*/
```

```
class UniformRng : public RngBasis {

public:

/// GROUP: Constructor and destructor

    // If no seed is specified, use the system clock as seed.
    UniformRng(unsigned long seed = 0);

    virtual ~UniformRng();

/// GROUP Methods

    virtual double rand();

    // Initialize the table.
    virtual void init();

private:

    unsigned long table_[16384];
    int index_;     // index into the table

};

#endif _UniformRng_h_


// UniformRng.cpp -- implementation file for the class UniformRng

#include "UniformRng.h"

/// GROUP: Constructor and destructor
UniformRng::UniformRng(unsigned long seed)
  : RngBasis(seed)
{
    init();
}

UniformRng::~UniformRng()
{}
```

```
/// GROUP: Methods

void UniformRng::init()
{
    unsigned seed = seed_;
    for (unsigned i = 0; i < 16384; i++) {
        table_[i] = seed = parkMiller(seed);
    }
    index_ = 0;
}

double UniformRng::rand()
{
    index_ = (index_ + 1) & 16383 ;
    // 15913 = 16384 - 471,   14798  = 16384 - 1586
    // 9396  = 16384 - 6988,  6695   = 16384 - 9689
    register unsigned long temp =   table_[(index_ + 15913 ) & 16383]
                                  ^ table_[(index_ + 14798 ) & 16383]
                                  ^ table_[(index_ + 9396  ) & 16383]
                                  ^ table_[(index_ + 6695  ) & 16383];
    table_[index_] = temp;
    return temp / m32Double_;// ((double) temp) / m32Double_;
}
```

Bibliography

[1] Arthur W.B.: *Inductive Reasoning and Bounded Rationality (The El Farol Problem)*, American Economic Review (Papers and Proceedings), 84, pp.406-411 (1994)

[2] Bank for International Settlements: *Triennial Central Bank Survey: Foreign exchange and derivatives market activity in 2001*, www.bis.org (March 2002)

[3] Barndorff-Nielsen O.E.: *Exponentially Decreasing Distributions for the Logarithm of Particle Size*, Proceedings of the Royal Society London A, 353, pp.401-419 (1977)

[4] Barndorff-Nielsen O.E.: *Processes of Normal Inverse Gaussian Type*, Finance and Stochastics, 2, pp.41-68, (1998)

[5] Barndorff-Nielsen O.E. and Halgreen O.: *Infinite Divisibility of the Hyperbolic and Generalized Inverse Gaussian Distributions*, Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete, 38, pp.309-312, (1977)

[6] Black F. and Scholes M.: *The Pricing of Options and Corporate Liabilities*, Journal of Political Economy, 81, pp.637-659, (1973)

[7] Bollerslev T.: *Generalized Autoregressive Conditional Heteroscedasticity* , Journal of Econometrics 31, pp.307-327, (1986)

[8] Breymann W.: *Dynamic Theta Time: Algorithm, Configuration, Tests*, Internal Document WAB.2000-07-31, Olsen&Associates, Seefeldstrasse 233, 8008 Zürich, Switzerland (2000)

[9] Breymann W., Ghashghaie S. and Talkner P.: *A Stochastic Cascade Model for FX Dynamics*, International Journal of Theoretical and Applied Finance, pp.357-360 (2000)

[10] Challet D., Chessa A., Marsili M. and Zhang Y.-C.: *From Minority Game to Real Markets*, http://xxx.lanl.gov/archive/cond-mat, arXiv:cond-mat/0011042, November 2, (2000)

[11] Challet D., Marsili M. and Zecchina R.: *Statistical Mechanics of Systems with Heterogeneous Agents: Minority Games*, http://xxx.lanl.gov/archive/cond-mat, arXiv:cond-mat/9904392 v2, November 29, (1999)

[12] Challet D., Marsili M. and Zhang Y.-C.: *Stylized Facts of Financial Markets and Market Crashes in Minority Games*, http://xxx.lanl.gov/archive/cond-mat, arXiv:cond-mat/0101326, January 22, (2001)

[13] Challet D. and Zhang Y.-C.: *On the Minority Game: Analytical and Numerical Studies*, http://xxx.lanl.gov/archive/cond-mat, arXiv:cond-mat/9805084 v2, May 8, (1998)

[14] Dacorogna M. Gençay R., Müller U., Olsen R. and Pictet O. *An Introduction to High-Frequency Finance*, Academic Press, pp.209-210 (2001)

[15] Dacorogna M. Gençay R., Müller U., Olsen R. and Pictet O. *An Introduction to High-Frequency Finance*, Academic Press, p.127 (2001)

[16] Eberlein E. and Keller U.: *Hyperbolic Distributions in Finance*, Bernoulli, 1, pp.281-299 (1995)

[17] Engle R.F.: *Autoregressive Conditional Heteroscedastic Models with Estimates of the Variance of United Kingdom Inflation*, Econometrica 50, pp.987-1007, (1982)

[18] Frisch U.: *Turbulence*, Cambridge University Press, (1995)

[19] Ghashghaie S., Breymann W., Peinke J., Talkner P. and Dodge Y.: *Turbulent Cascades in Foreign Exchange Markets*, Letters to Nature, Vol. 381, pp.767-770 (1996)

[20] Goodhard C. and O'Hara M.: *Introductory Lecture: A Survey of High Frequency Micro-Market Structure Studies*, First International Conference on High Frequency Data in Finance Zurich, Switzerland, March 29-31, (1995)

[21] LeBaron B.:*A Builder's Guide to Agent-Based Financial Market*, Quantitative Finance, Vol. 1, pp.254-261, (2001)

[22] Lux T. and Marchesi M.: *Scaling and Criticality in a Stochastic Multi-Agent Model of a Financial Market*, Letters to Nature, Vol. 397, pp.498-500 (1999)

[23] McNeil A.J.: *EVIS Software for Extreme Value Theory*, Departement Mathematik, ETH Zentrum, CH-8092 Zürich or www.math.ethz.ch/∼mcneil

[24] Merton R.C.: *Theory of Rational Option Pricing*, Bell Journal of Economics and Management Science, 4, pp.141-183, (1973)

[25] Müller U.A., Dacorogna M.M., Davé R.D., Olsen R.D., Pictet O.V. and von Weizsäcker J.E.: *Volatilities of Different Time Resolutions - Analyzing the Dynamics of Market Components*, Journal of Empirical Finance 4, pp.213-239, (1997)

[26] Olsen & Associates, Research Institute for Applied Economics, Seefeldstrasse 233, 8008 Zürich, Switzerland; www.olsen.ch

[27] The World Bank Group, www.worldbank.org

[28] Zhang Y.-C. and Challet D.: *Emergence of Cooperation and Organization in an Evolutionary Game*, http://xxx.lanl.gov/archive/cond-mat, arXiv:adap-org/9708006v2, September 3, (1997)

[29] Zumbach G. and Müller U.: *Operators on Inhomogeneous Time Series*, International Journal of Theoretical and Applied Finance, 4(1), pp.147-178 (2001)

Acknowledgements

It is a great pleasure to thank Prof. Dr. Jürg Hüsler for proposing the problem of Part I "Extreme Values of Gaussian Processes". I enjoyed many inspiring scientific discussions and I am grateful for his administrative support.

PD Dr. Wolfgang Breymann from the Risklab at ETH Zürich was the FX specialist of Part II "A Heterogeneous Multi Agents Model". I am thankful for receiving insights into the philosophy of modeling financial markets and for his organization of my research stay at the "Zentrum für interdisziplinäre Forschung" in Bielefeld.

I thank Dr. Shoaleh Ghashghaie for her tips during my first steps in the area of finance.

Olsen company provided time series of real FX rates, which I very appreciate.

Many special thanks to Dr. Manuel Walker for his excellent advice concerning C++. His huge experience with computers were also helpful in many other situations.

Last but not least I would like to thank my fellow students of the "Institute of Mathematical Statistics and Actuarial Sciences" for their hints concerning statistics, which was for me, coming from theoretical physics, very helpful. Especially I thank Yuexiang Jiang, Manuel Bertschy and René Burkhard for sharing not only the office, but also their preference for electronic music while working.

Curriculum Vitae

Personal Data

Name	Schmid, Christoph Manuel
Date of Birth	September 3, 1974
Place of Birth	Bern, Switzerland
Nationality	Swiss
Marital Status	unmarried
Present address	Institut für mathematische Statistik und Versicherungslehre Universität Bern Sidlerstrasse 5 CH-3012 Bern E-mail: `christoph.schmid@stat.unibe.ch`

Schools

1981-1990	Schools in Fahrni and Unterlangenegg
1990-1994	Gymnasium Thun
1994	Matura Typus C (University Entrance Exam)

University

1994-1999	University of Bern Studies in physics, mathematics and astronomy
1999	University of Bern MSc in theoretical physics, "Magnetic Monopoles in $N = 2$ Supersymmetric $SU(2)$ Yang Mills Theory" (supervised by Prof. Dr. Peter Minkowski)
1999-2002	University of Bern PhD in mathematical statistics and actuarial sciences "Extreme Values of Gaussian Processes and A Heterogeneous Multi Agents Model" (supervised by Prof. Dr. Jürg Hüsler)

Other Activities

1994	Obligatory Military Service
2001	Zentrum für Interdisziplinäre Forschung Bielefeld, Germany
2001	London School of Economics, England "Summer School on Options, Futures and other Financial Derivatives"