Enhanced Forecasting Methods, Fat Tails, and their Applications in Finance

Zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

von der Fakultät für
Wirtschaftwissenschaften
des Karlsruher Institut für Technologie

genehmigte

Dissertation

von

Dipl.-Phys. Christian Scherrer

Tag der mündlichen Pruefung: 20.12.2010
Referent: .....Prof. Dr. S.T. Rachev
Korreferent: .....Prof. Dr. M. Feindt

Erklärung

Ich versichere wahrheitsgemäß, die Dissertation bis auf die in der Abhandlung angegebene Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und genau kenntlich gemacht zu haben, was aus Arbeiten anderer und aus eigenen Veröffentlichungen unverändert oder mit Abänderungen entnommen wurde.

Contents

List of Figures

List of Tables

1

Introduction

This work originates from a collaboration with Phi-T®, a signal provider for the Lupus alpha Neurobayes Short Term Trading fund. This fund is managed using various automatic trading strategies. The thesis presented here discusses two different research topics which are important in the portfolio management.

In the first part[1] we deal with the key task in asset management, namely to estimate and forecast the risk of an asset. Risk is here defined in its most general form as 'the probability that something special happens'. The details will be discussed below. Therefore we have to find a mathematical model which is capable of forecasting the probability density function of asset returns. As pointed out in Bollerslev (1986a) and Engle (1982a), the most common models belong to the class of ARMA-GARCH models. An ARMA-GARCH model basically assumes a stationary stochastic process. The asset return at time t is described as depending on previous asset returns and in addition, the volatility of the returns varies slowly in time. Furthermore, exogenous variables can be included in the model. In this case one has to assume a functional relationship between asset return and exogenous variables. Since this functional relationship can be arbitrarily complicated, the most common models adopt a linear relationship. An ARMA-GARCH model has the great advantage that it is easy to define within the framework of arbitrage pricing theory. Therefore a risk-free process can be constructed which can be used to compute prices of derivatives such as options and futures.

One objective of this work is the comparison of a neural network model with the class of ARMA-GARCH models. Furthermore, it should be possible to define a risk-free process. In this case it is possible to compute prices of derivatives in the framework of arbitrage pricing theory. Neural networks are capable of learning the empirical conditional probability

[1] This part is discussed in Scherrer *et al.* (2010b)

density function of a so-called 'target' (e.g. asset returns) using historical data. The target depends on a multi-dimensional vector of input variables in an arbitrary nonlinear way. Therefore it is not necessary to assume a fixed distribution or a stationary stochastic process describing the asset returns. A second advantage is that we can easily include arbitrary information in the input variables of the neural network. Therefore the model can easily be extended. Moreover, if we believe to have found an input variable that is correlated to the target in an arbitrary nonlinear way, a neural network will find this relationship. If the input variable is meaningless for the variable, the neural network will ignore it.
In the first part of this work we introduce two neural network models which are compared to three types of ARMA-GARCH models that have already been discussed in literature (see Bollerslev (1986a), Engle (1982a), Kim *et al.* (2009), Bollerslev (1987)).
The second part of the thesis introduces a very fast algorithm that is refered to as 'individualized semi-linear regression' and discussed in Scherrer *et al.* (2010a). A simple linear regression is one of the most important tools and widely used in finance. The algorithm is basically a linear regression in which slope and intercept depend on regimes that are defined with the help of exogenous variables. With this in mind the results of a simple linear regression can be improved dramatically. It is assumed that slope and intercept vary continuously across the regimes. The algorithm is capable of dealing with correlated inputs and identifies only statistically significant correlations of exogenous variables to the parameters of the regime shifted linear regression. The algorithm is robust with respect to statistical outliers under fairly general conditions.
The motivation for the individualized semi-linear regression is based on an analysis of the market impact as a trader. When an investor trades an asset at an exchange, the prices of the asset will change. In general, when the investor buys an asset, the price will increase, while it decreases when he or she sells the asset. The size of the price change depends statistically significantly on the size of the order and appears to be approximately linear in order-size. The individualized semi-linear regression was developed to analyse higher order corrections to the linear market impact model. The market impact analysis is discussed in Fraenkle *et al.* (2010) and is not included in this thesis.
Technical details and the mathematical background can be found in the appendix.

Part I

Backtesting Risk methodologies

2

Introduction

A key task of risk managers and asset managers is estimating and forecasting the risk of an asset. One of the most common approaches for doing so is the mathematical modeling of a time series with help of a generalized autoregressive conditional heteroscedasticity (GARCH)-type model. After choosing the type of model one has to employ statistical tests to assess if the historical data can be described accurately by the selected model. The next step is to perform a backtest (i.e., a simulation) using all information available until day i to forecast the next daily return. Since the real return for an asset is known, it is possible to compare the prediction and the actual value and then decide if the model is sufficiently reliable so as to be useful in predicting future returns. This test is necessary to exclude trivial models which have enough parameters to learn a complete historical data sample but which are not capable of real forecasting.

Many studies have shown that the assumption of a normal distribution for the residuals of a GARCH-model is inappropirate because asset returns are generally skewed and have a excess kurtosis (see, among others, Menn and Rachev (2005a) and Menn and Rachev (2005b)). An alternative to a Gaussian is the class of tempered stable distributions. The **c**lassical **t**empered **s**table (CTS) distribution will be the distribution used in this paper. The definition of this type of distribution is given in Rosinski (2007a). The CTS has been used for the residuals of a GARCH model in Kim *et al.* (2008a) and Kim *et al.* (2008c). The mathematical details with respect to the CTS distribution, needed in this work, are summed up in C.6.3.2.

We propose five models and compare their performance using a large backtest in which the daily returns of the Dow Jones Industrial Average (DJIA) are predicted from 1987 until

2009. We employ as our benchmark a normal-ARMA-GARCH and a t-ARMA-GARCH model to define a process for the daily returns. Additionally, we use a CTS-ARMA-GARCH model which has the advantage of describing the skewness and fat tails that has been observed for assets in numerous studies. The remaining two models are two neural network models which take into account volatility clustering, another stylized fact observed about asset prices. The class of ARMA-GARCH models as well as the neural network models are briefly described in the appendix.

Originally, neural networks have been used only for classification problems. In this case, the network is given a data sample in which for a certain event i the input vector $\overrightarrow{x_i}$ and the target t_i are specified. The target can be a signal ($t_i = 1$) or a background ($t_i = -1$). The network is then trained with these samples. In other words, the network can learn and get experience from examples and afterwards it can be used to predict the probability of an event being a signal under the condition that the input vector is given as $\overrightarrow{x_i}$.

Neural networks are especially useful when there is a large number of historical events in which an input vector and a target variable are provided. They can approximate any universal function by an arbitrary grade of acuracy (see among others Irie and Miyake (1988), Cybenko (1989), Funahashi (1989), Hornik (1989), Hartman *et al.* (1990)). Therefore it is possible to learn nonlinear correlations between input variables and target. Although neural networks were originally applied to classification problems, the main obstacle with a time series of daily returns is that we cannot define a return as a signal or a background. This is due to the fact that returns are continuous rather than a binary decision. In Weigend and Srivastava (1995) and Feindt (2004), this problem was solved using several output nodes. While in Weigend and Srivastava (1995) the output nodes have been used to model the probability density function (pdf), in Feindt (2004) the cumulative distribution function (cdf) was fitted. With this improvement, it is possible to predict a conditional probability density function for the return of an asset given some information at day i. The output of the neural network can be given by random numbers which are generated from the pdf. These random numbers are fitted by a CTS distribution which is used to define a stochastic process. If one has defined the process, a risk-neutral process can be found so that the model can be used within the framework of arbitrage pricing theory, see Kim *et al.* (2008a).

3

Definition of the neural network models

The goal in designing a risk management methodology using a neural network is to be able to predict the probability density function for the daily DJIA return $f(r_i|\overrightarrow{x_i})$. To do so requires several steps which we discuss here.

3.1 Definition of the target

First, a target variable for the network must be defined. It is easier for the network to learn a distribution which does not have extreme outliers. Therefore, the idea is not to use directly the return as a target variable. Instead, a transformed return must be computed. That means if we would like to predict the return, we would get the density of the transformed return from the neural network. Then a back transformation from this target variable must be computed in order to obtain the return again.

The transformation just normalizes the daily returns. This is done by dividing the daily returns by volatility. So we have

$$\tilde{r}_i = \log\left(\frac{\text{close}_{i+1}}{\text{close}_i}\right)\frac{\sqrt{253}}{\sigma_i} \tag{3.1}$$

where σ_i is an estimate for the yearly volatility derived from the daily history of the DJIA, including the close of day i. Failure to normalize the returns means that the neural network would also have to learn the volatility clustering, which is not easy.

It is important to note that σ_i only includes information up to day i which means that no information of the future is included. If we had included information of the future in the

volatility, we would have had to predict the distribution of the return and the distribution of the volatility. That means it is best to define a target which has similar properties for each and every day but which does not include more than one variable which embodies future information.

The target is then defined by

$$t_i = \tilde{r}_i \tag{3.2}$$

For the backtest we will use two different neural network models which will only differ in the definition of the target:

1. *EWMA-CTS-nn*: Neural network with exponentially weighted moving average volatility[1] in the definition of the target
2. *GARCH-CTS-nn*: Neural network with historical volatility from normal-ARMA-GARCH in the definition of the target

3.2 Definition of the input vector

In the second step the input vector from which the network can learn the conditional probability density function must be defined. We take into account for the daily returns of the DJIA the open, high, low, and close, and from these data we construct input variables for every trading day. For example at day i complete information of the past up to day i can be used. That means the latest information we use is given by the close, open, high, and low of day i. From the time series up to day i we construct 51 input variables for the neural network. These input variables include many well-known technical indicators which can be found in Colby (2002) and Achelis (2000). In addition, we also use variables we constructed (e.g., coefficients of wavelets or combined variables of technical indicators).[2] The input variables for the neural network models computed by using different time series algorithms of Phi-T® are:

- 14 variables from wavelet analysis

[1] We use a decay constant of 14 days.

[2] The variables are used in a commercial model at Phi-T® and cannot be described in detail due to their proprietary nature.

- 12 variables constructed from combinations of $high_i$, low_i, $close_i$, $open_i$, $high_{i-1}$, low_{i-1}, $close_{i-1}$, $open_{i-1}$
- 6 variables constructed from $close_i$, ..., $close_{i-k}$
- 6 different volatilities (using moving averages, exponential moving averages, Wilder's volatility)
- 4 relative strength indices defined on different time intervals
- 4 different combinations of moving averages defined on different time intervals
- 2 variables which include different stochastic oscillators
- 1 variable using Bollinger bands
- 1 variable constructed from the Moving Average Convergence/Divergence indicator
- 1 variable using Williams %R indicator

3.3 How to forecast daily returns

After defining the input vector and the target, we can train our network and try to predict future returns. We would like to have a prediction for

$$r_i = \log\left(\frac{\text{close}_{i+1}}{\text{close}_i}\right) \tag{3.3}$$

That means close_{i+1} is future information and close_i is known. But we also know high, low, and open of day i. That means we can use all data in our sample up to day i to train the network and adjust its weights. Then we compute the input vector $\overrightarrow{x_i}$. We insert this input vector into the network and get a forecast for t_i which is defined in (3.2).

The NeuroBayes software[3] is able to predict random numbers of t_i, so we have to do the back transformation of (3.1) to get the density for r_i. So we use NeuroBayes to obtain random numbers of the returns.

We would like to operate in the framework of the Arbitrage Pricing Theorem (APT) in order to guarantee that the model also works for the pricing of derivatives. We can make use of proposition (2.2) from Kim *et al.* (2009), which states that an equivalent martingale exists if we can define a stochastic CTS process. The discretized version of this CTS process

[3] Developed by Phi-T® Physics Information Technologies GmbH

will be driven by random increments, generated by NeuroBayes[4].

We focus on testing the forecasting capabilities of different models in this work, so we do not mathematically introduce the equivalent martingale. Basically, the APT states that if we define any payoff function of a derivative and if there exists a strategy which replicates this payoff function, the prices of the derivative and the strategy have to be the same. Otherwise one could trade the strategy against the derivative and would realize a risk-free gain. If we have a measure P and we can define an equivalent measure Q under which the discounted stochastic process is a martingale, the fair price of the derivative is defined by the expected payoff under the equivalent martingale measure. This is essentially a consequence of the martingale representation theorem, see Protter (2003).

The CTS distribution has six free parameters (see C.6.3.2) and the fit is numerically difficult because we have to solve a highly nonlinear optimization problem. Therefore we normalize the random numbers using the transformation

$$\tilde{r}_i = \frac{r_i - \mu_i}{\sigma_i}$$

where i denotes the day in the time series, μ_i is the mean of the random numbers, and σ_i is the root mean square.
Now we can fit the stdCTS probability density function to the transformed random variables r_i, so we have to estimate the parameters $(\tilde{\alpha}, \tilde{\lambda}_+, \tilde{\lambda}_-)$. Using transformation (3.3) we have three degrees of freedom instead of six degrees of freedom in the optimization problem.

For the fit of the CTS distribution one has to be aware of the problem that from the CTS distribution we know just the characteristic function, i.e. we have to compute the inverse Fourier transform. Therefore we use the Fast Fourier Transform (FFT)[5] to compute the pdf of the CTS distribution numerically. The fit is done using the Maximum Likelihood Method, discussed in chapter A in the appendix. For the Maximum Likelihood estimation we tried two different nonlinear programming algorithms (NLP). In the first algorithm, we used the function fmincon which is a standard function in Matlab. In the second algorithm,

[4] The transition between the discrete and the continuous space is not in the scope of our discussion and can be found e.g. in Billingsley (1968)
[5] The FFT requires $O(n\log(n))$ operations instead of $O(n^2)$ operations in case of the discrete Fourier transform.

we implemented the problem using Knitro[6]. We compared the two algorithms and found that the likelihood of the solutions using Knitro are better. Therefore all parameter estimations are implemented with the Knitro solver.

After estimating the three parameters, we get the parameters of the non-normalized CTS distribution using

$$\begin{aligned} \alpha &= \tilde{\alpha} \\ C &= \sigma^{\tilde{\alpha}} \tilde{C} \\ \lambda_+ &= \frac{\tilde{\lambda}_+}{\sigma} \\ \lambda_- &= \frac{\tilde{\lambda}_-}{\sigma} \\ m &= \mu \end{aligned} \tag{3.4}$$

where $\tilde{C}$ is defined in (C.12) using the parameters $(\tilde{\alpha}, \tilde{\lambda}_+, \tilde{\lambda}_-)$. The proof can be found in Scherer *et al.* (2010) which uses the definitions in Kim *et al.* (2008b).

[6] www.ziena.com/knitro.htm

4

Definition of the used ARMA-GARCH models

In chapter C in the appendix we discuss the class of ARMA-GARCH models and explain that there are different possibilities for the distribution of the residuals. We include the three following models in our backtest:

1. *normal-ARMA-GARCH*: ARMA(1,1)-GARCH(1,1) model with standard normal distributed innovations
2. *t-ARMA-GARCH*: ARMA(1,1)-GARCH(1,1) model with t-distributed innovations
3. *CTS-ARMA-GARCH*: ARMA(1,1)-GARCH(1,1) model with classical tempered stable distributed innovations

The normal-ARMA-GARCH and the t-ARMA-GARCH are the most common models in financial time series and we use them in order to have a benchmark. We use the Matlab toolbox for the estimation of the parameters of these models.

The CTS-ARMA-GARCH model is not directly implemented in Matlab. Therefore we implemented a method which is based on the following steps:

1. Estimate the parameters of the normal-ARMA-GARCH model.
2. Compute the residuals using the parameters coming from the normal-ARMA-GARCH model.
3. Fit a classical tempered stable distribution to the residuals.

Fitting the parameters is far away from being trivial. If we fitted all parameters in one step using the Maximum Likelihood Method (see chapter A), we would not find a numerically stable solution, because we had a highly nonlinear optimization problem and many correlated parameters to estimate.

Therefore we use the Quasi-Maximum-Likelihood Method to estimate the parameters in a two-step method where we fit a normal-ARMA-GARCH model to the timeseries and use the residuals to estimate the parameters of the CTS distribution in a second step.

A second problem would occur in a one-step method. The volatility clustering is used to be modeled by a GARCH model. But if we used a one-step procedure, the effect of volatility clustering would be absorbed by the CTS distribution.

The first two steps are straightforward using the Matlab toolbox. The residuals are fitted with the same method used to fit the output of the neural network (see chapter 3.3).

5

The Backtest

5.1 How to define a reasonable backtest

In backtesting our model, we encounter the problem of forecasting a probability density function for one event while having only one actual realization of that event. The forecasted density is dependent on the input vector which we insert in the neural network which again depends on day i. That means that the density itself is also dependent on day i.

One generally accepted possibility to check the forecasted densities is to count exceedences of the return with respect to a specific quantile. In this case the most common choices for the quantile are 1% or 5%. The idea is then that in an infinitely large sample, the exceedences of the actual return with respect to the 1% quantile of the forecasted distribution should converge to the actual probability (i.e., to 1%). But if we only looked at this quantile, we would fail to be using most of the statistics. So it would be much more reasonable to look at many quantiles.

When counting the exceedences what we basically do is forecast the density of an event and measure its actual return of it. Then the cdf of the actual return which is defined in the interval $[0, 1]$ can be computed. The pdf is dependent on the event but the cdf for the actual returns z_i should always be a uniform distribution independent of the chosen event; that is,

$$z_i = \int_{-\infty}^{r_{actual}^{(i)}} dr \; f(r|\overrightarrow{x_i}) \quad z_i \in [0, 1] \tag{5.1}$$

Therefore, it is reasonable to check if the cdf of the actual event results in a uniform distribution (if all quantiles of our forecasted distribution are correct). For a further discussion,

see Campbell (2006) and the references therein.

So far we explained how we can test if our density is correct, but we still have not quantified how we really measure whether the cdf of the actual returns is uniform and how we can compare different models. For this purpose, we turn to the Kolmogorov-Smirnov and the Anderson-Darling tests. The basic idea in both tests is the same. One involves computing the theoretical cdf of the actual returns and the empirical cdf and then takes the maximal distance between them. This distance is a random variable on which one can decide if the model is accepted or rejected. The Anderson-Darling test works in the same way but one assigns a higher weight to the correctness of the tails.

The same idea is improved upon in a methodology suggested by RiskMetrics (see Zumbach (2006)). The methodology involves first introducing the variable

$$\delta(z) = \mathrm{cdf}_{emp.}(z) - z \qquad z \in [0,1] \tag{5.2}$$

which is the difference between the empirical and the theoretical cdfs. If the model is correct, $\delta(z)$ should converge to 0 for all values of z for an infinitely large sample. So one possibility to compare different models is to plot $\delta(z)$ with the better model being the one with the smaller absolute values for the deltas.

The second suggestion by RiskMetrics is to construct a scalar from the computed deltas which is a measure for the correctness of the entire cdf. For this purpose the variable d_p is introduced where p is a parameter with which different weight can be given to the tails

$$d_p = \int_0^1 dz \mid \delta_p(z) \mid \tag{5.3}$$

and δ_p is given by

$$\delta_p(z) = \delta(z)(p+1)2^p \mid z - \frac{1}{2} \mid^p \tag{5.4}$$

If $p = 0$, this is just the integral over all absolute values of $\delta(z)$. The larger the computed p, the greater the importance of describing the tails accurately.

5.2 Approach

First, we downloaded all available data (1930-2009) for the time series for the Dow Jones Industrial Index (DJIA) from www.finance.yahoo.com. For the forecast we chose the years from 1987 until 2009. Our selection of 1987 was because it included Black Monday (October 19, 1987), the largest one day decline in the DJIA in stock market history since 1929.

We use all data up to the last day in 1986 and train our neural network to adjust the parameters. We predict the pdf for the first return in 1987 and compute the cdf of the actual return as described in the previous section. The next step is to predict the second return. In principle, we now have one more event in the historical data (the first return in 1987). That means we could train the network again with the same data sample as in the first training plus this event. Since the computational effort would be quite large if we did a new training for each trading day, we decided to compromise by doing a new training after one month has passed. That means to predict January 1987, we use a training which includes data from 1930 until December 1986. After this month, we train the network again with data from 1930 until the end of January 1987 and so on.

The backtest is done for the following five models:

1. *normal-ARMA-GARCH*: ARMA(1,1)-GARCH(1,1) model with standard normal distributed innovations
2. *t-ARMA-GARCH*: ARMA(1,1)-GARCH(1,1) model with t-distributed innovations
3. *EWMA-CTS-nn*: Neural network with exponentially weighted moving average volatility in the definition of the target and a fit of a CTS-distribution to the output of the network
4. *GARCH-CTS-nn*: Neural network with historical volatility from normal-ARMA-GARCH in the definition of the target and a fit of a CTS-distribution to the output of the network
5. *CTS-ARMA-GARCH*: ARMA(1,1)-GARCH(1,1) model with classical tempered stable distributed innovations

In the GARCH-models, the computational costs are low and the estimation of the parameters is done for every trading day.

5.3 Results of the backtest

In Section 5.1 we explained that the distribution of the cdf of the actual returns should be a uniform distribution if the model were acceptable. Therefore, we compare these plots for our different models. In Figures 5.1 and 5.2 we can see the results for the normal-ARMA-GARCH model and the t-ARMA-GARCH model. As expected, the cdf of the tails of the actual returns is much heavier than that predicted by the normal distribution. In particular, the large losses are not described very well. In the central area there are more events than described by the normal distribution.

In case of the t-ARMA-GARCH model, the tails (especially the right tails) seem to be too fat while the central area is underestimated as observed for the normal-ARMA-GARCH model. The missing property of both models is skewness. But assets exhibit the typical property of being skewed to the left.

The results of the neural network models are presented in Figures 5.3 and 5.4. We can see immediately that the distributions are quite uniform and that the central area and the fat tails are described well. The asymmetry which can be seen in the plots of the two GARCH models studied does not appear in the neural network models.

In case of the CTS-ARMA-GARCH model (see Figure 5.5), the cumulative distribution of the actual return is also quite uniform and the tails and the asymmetry are described well.

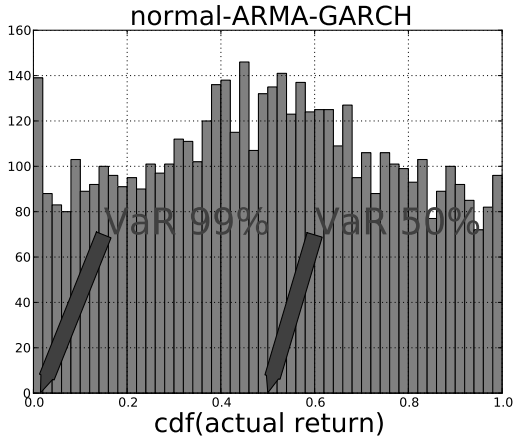

Fig. 5.1: Cumulative distribution function of the actual return using the normal-ARMA-GARCH model.

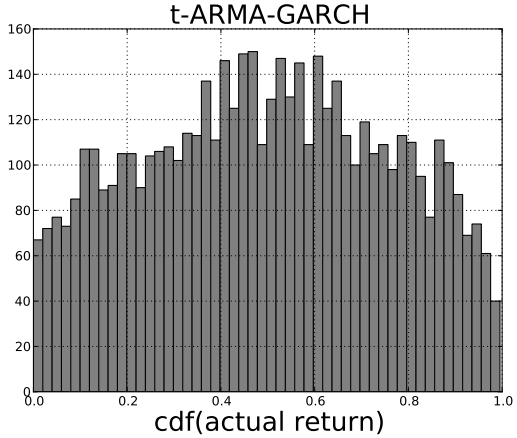

Fig. 5.2: Cumulative distribution function of the actual return using the t-ARMA-GARCH model.

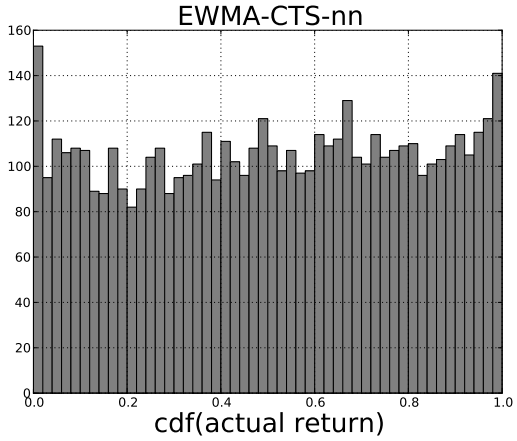

Fig. 5.3: Cumulative distribution function of the actual return using the EWMA-CTS-nn model.

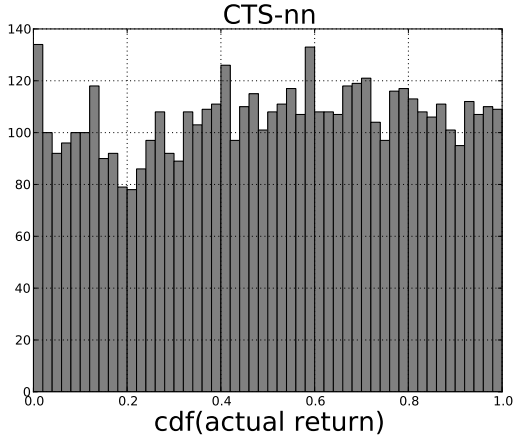

Fig. 5.4: Cumulative distribution function of the actual return using the GARCH-CTS-nn model.

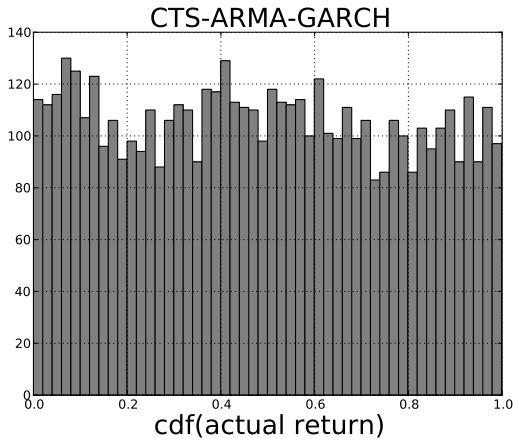

Fig. 5.5: Cumulative distribution function of the actual return using the CTS-ARMA-GARCH model.

The quantile-quantile-plots of the cdf distribution of the actual return are presented in the Figures 5.6, 5.7, 5.8, 5.9 and 5.10. Again we can see that the asymmetry is a very important missing property of the normal-ARMA-GARCH and the t-ARMA-GARCH model.

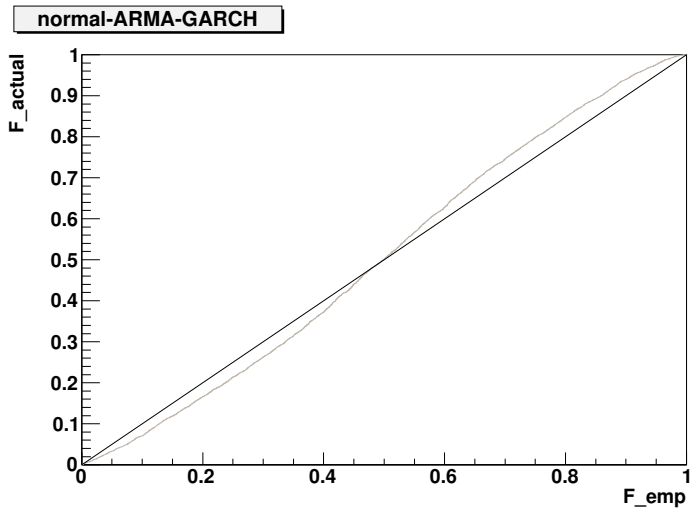

Fig. 5.6: QQ-plot of the cumulative distribution function of the actual return using the normal-ARMA-GARCH model.

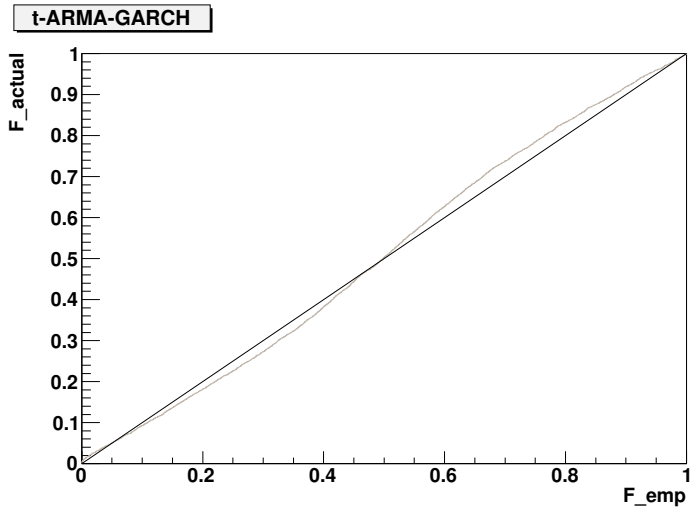

Fig. 5.7: QQ-plot of the cumulative distribution function of the actual return using the t-ARMA-GARCH model.

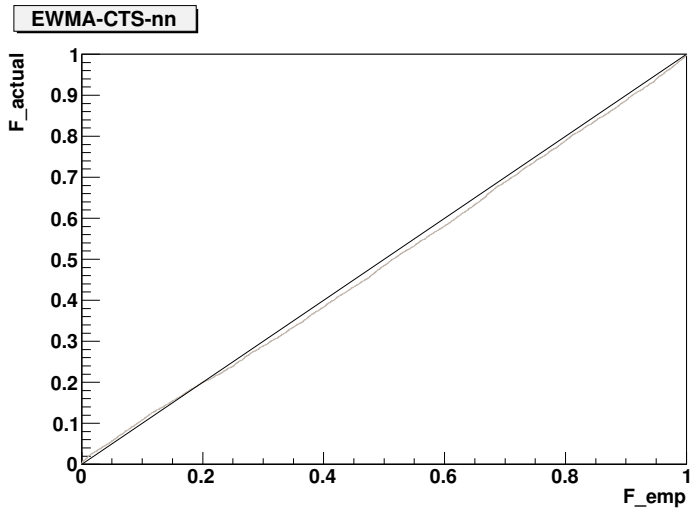

Fig. 5.8: QQ-plot of the cumulative distribution function of the actual return using the EWMA-CTS-nn model.

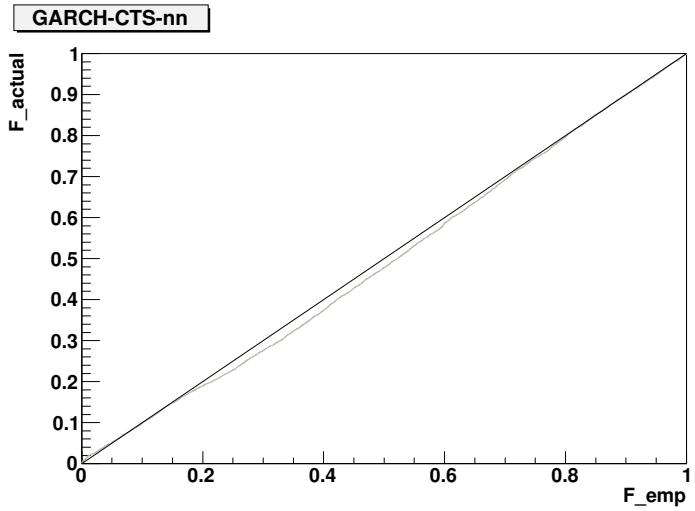

Fig. 5.9: QQ-plot of the cumulative distribution function of the actual return using the GARCH-CTS-nn model.

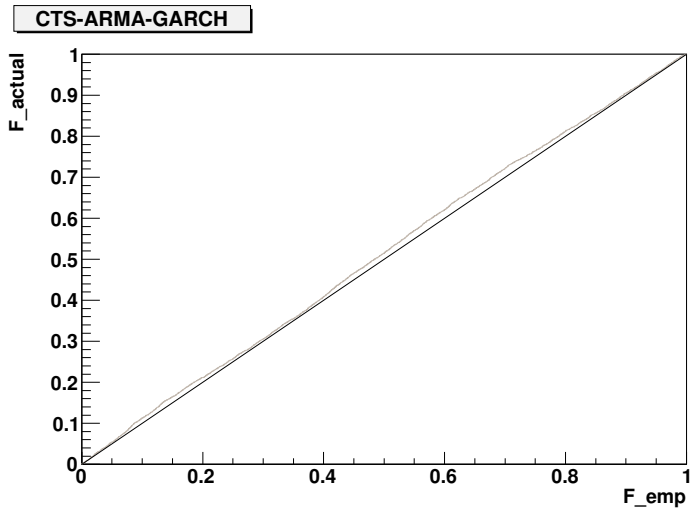

Fig. 5.10: QQ-plot of the cumulative distribution function of the actual return using the CTS-ARMA-GARCH model.

Next we compare the δ_p's, defined in (5.4) to quantify the forecasting abilities of the different models. For $p = 0$, no special weight is given to the tails and we end up with Figure 5.11. These are just the differences between the theoretical and empirical cdfs. Again we see that the normal-ARMA-GARCH and the t-ARMA-GARCH models are comparable while both neural network models and the CTS-ARMA-GARCH model are much better because the absolute differences between the theoretical and the empirical cdf are much smaller.

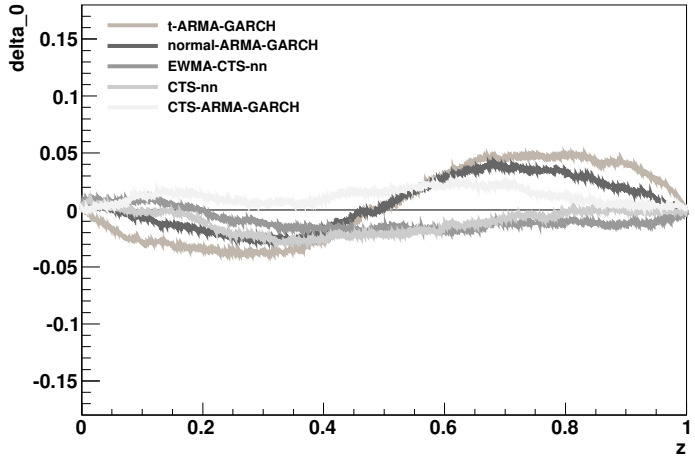

Fig. 5.11: Difference between empirical cdf and theoretical cdf for all models (giving no special weight to the tails).

To investigate the tail properties of the models, we follow RiskMetrics and plot δ_p defined in (5.4) for $p = 32$ (see Figures 5.12 and 5.13). Again we see that the left tail of the normal-ARMA-GARCH model is not heavy enough while the t-ARMA-GARCH model has a left tail that is too heavy. For the left tail the CTS-ARMA-GARCH model is the best one while the right tail is described best by the GARCH-CTS-nn model.

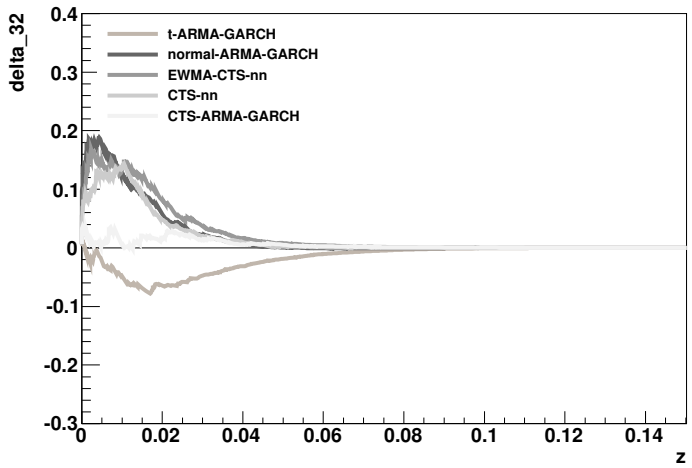

Fig. 5.12: Difference between empirical cdf and theoretical cdf for all models (giving more weight to the tails).

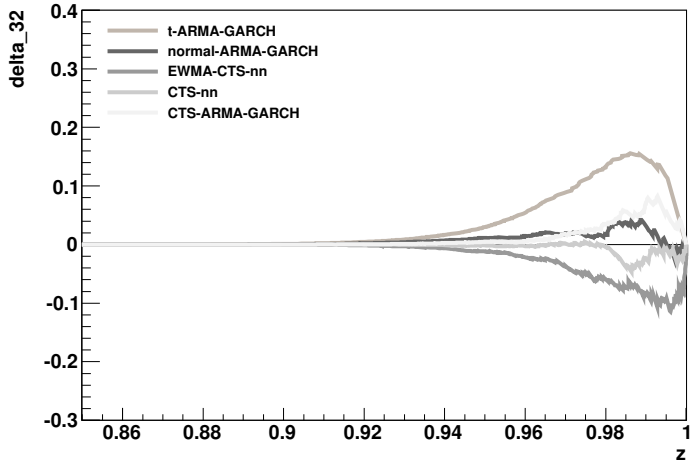

Fig. 5.13: Difference between empirical cdf and theoretical cdf for all models (giving more weight to the tails).

	normal-ARMA-GARCH	t-ARMA-GARCH	EWMA-CTS-nn	GARCH-CTS-nn	CTS-ARMA-GARCH
d_0	0.0191	0.0300	0.0128	0.0125	**0.0101**
d_{32}	0.0042	0.0071	0.0068	0.0036	**0.0022**

Table 5.1: Comparison of the performance of all models.

In order to have a scalar which characterizes the complete model we followed again RiskMetrics and computed d_0 and d_{32} as defined in (5.3). The results are reported in Table 5.1. Note that d_0 and d_{32} are both random numbers which include a statistical uncertainty. Again, we see that both neural network models and the CTS-ARMA-GARCH model clearly outperform the normal-ARMA-GARCH and the t-ARMA-GARCH models. The central area is similar in both neural network models and the CTS-ARMA-GARCH model, while the tails are described best in the GARCH-CTS-nn model and the CTS-ARMA-GARCH model. The GARCH-CTS-nn model is even slightly better than the EWMA-CTS-nn model.

In order to see how well the models perform in different time intervals, we computed d_0 and d_{32} for every year. The results are summarized in Table 5.2 and Table 5.3. The normal-ARMA-GARCH model is the worst performing model for the estimation of the tails while the t-ARMA-GARCH does not perform well in the central area. From this analysis we see that in general one should use advanced neural network models as well as advanced ARMA-GARCH models (such as CTS-ARMA-GARCH) to forecast the risk of an asset.

The markets in the years 2007 and 2008 were dominated by the financial crisis popularly referred to as the subprime mortgage crisis. Therefore, we plot the 1% value-at-risk (VaR) measure of all models in Figures 5.14 to 5.18. Especially in September and October 2008, the tail losses increased dramatically. The black dots symbolize the VaR violations. Again we see that the tails in the normal-ARMA-GARCH model are too thin while they are extremely fat in the t-ARMA-GARCH model. We summarize these violations in Table 5.4. The numbers of violations that are bolded are consistent with the 95% confidence interval of the Kupiec test (Kupiec (1995)). In this test we would reject the normal-ARMA-GARCH model for both years and both neural network models for the year 2007, while we would accept the t-ARMA-GARCH and the CTS-ARMA-GARCH model. A natural question to ask

is why the neural network models were not able to outperform the CTS-ARMA-GARCH model. To answer this question, one has to understand how the NeuroBayes software is able to create a pdf without assuming an analytical function. In principle, NeuroBayes reconstructs the cdf from a neural network with 20 output nodes. But using 20 nodes means that it is necessary to approximate the cdf between the nodes, adding further uncertainty to the model.

time span	normal-ARMA-GARCH	t-ARMA-GARCH	EWMA-CTS-nn	GARCH-CTS-nn	CTS-ARMA-GARCH
1987-1988	**0.022644**	0.030760	0.040212	0.040457	0.028297
1988-1989	0.038154	0.048520	0.027009	**0.018101**	0.020179
1989-1990	0.038065	0.058315	0.061605	0.046072	**0.026360**
1990-1991	0.032053	0.036725	0.017759	**0.014751**	0.021959
1991-1992	0.045607	0.058237	0.017688	0.019752	**0.016277**
1992-1993	0.054623	0.066664	**0.016332**	0.025612	0.028545
1993-1994	0.059484	0.066465	**0.026723**	0.043880	0.032445
1994-1995	0.016809	0.032303	**0.009143**	0.015154	0.016853
1995-1996	0.043439	0.048629	0.052760	0.051693	**0.026296**
1996-1997	0.022529	0.026427	0.035784	0.043713	**0.013983**
1997-1998	0.018159	**0.013125**	0.025526	0.041042	0.022264
1998-1999	**0.008944**	0.014877	0.012507	0.027169	0.018269
1999-2000	0.016726	0.020272	0.018360	**0.016350**	0.026765
2000-2001	0.024323	0.026312	0.027842	**0.018941**	0.028146
2001-2002	0.022391	0.026269	0.017002	**0.012332**	0.026585
2002-2003	0.041965	0.040762	0.043573	**0.035881**	0.051660
2003-2004	0.025904	0.034855	0.030652	0.025739	**0.013507**
2004-2005	0.033400	0.042242	0.017000	**0.016907**	0.032626
2005-2006	0.018641	0.023739	0.014251	**0.012594**	0.029929
2006-2007	0.032855	0.038259	0.029995	**0.022319**	0.025798
2007-2008	0.033232	0.034138	0.031309	0.033099	**0.024182**
2008-2009	0.034417	0.034999	0.026763	**0.025185**	0.042336

Table 5.2: Comparison of d_0 for all models depending on the time interval.

time span	normal-ARMA-GARCH	t-ARMA-GARCH	EWMA-CTS-nn	GARCH-CTS-nn	CTS-ARMA-GARCH
1987-1988	0.013496	**0.003051**	0.015146	0.015595	0.009155
1988-1989	0.006286	0.007846	0.021596	0.011474	**0.003703**
1989-1990	0.004091	0.008857	0.026887	0.017027	**0.002509**
1990-1991	0.008461	0.006354	0.008743	**0.005080**	0.007492
1991-1992	0.003730	0.010248	0.008351	0.006005	**0.002315**
1992-1993	0.009839	0.006559	**0.003821**	0.005877	0.007939
1993-1994	0.008969	0.004607	**0.003085**	0.007423	0.007329
1994-1995	0.006938	0.005760	0.007138	0.007500	**0.005153**
1995-1996	0.005370	0.008457	**0.003842**	0.006845	0.005599
1996-1997	0.006429	**0.006213**	0.010610	0.007321	0.008976
1997-1998	0.011416	**0.005615**	0.013973	0.013154	0.006351
1998-1999	0.008785	**0.004317**	0.005139	0.007104	0.004926
1999-2000	0.006034	**0.002746**	0.005220	0.003298	0.004898
2000-2001	0.011860	0.006267	**0.002795**	0.005773	0.007767
2001-2002	0.004968	0.007266	0.003313	0.003556	**0.003199**
2002-2003	0.007920	**0.002561**	0.009977	0.008295	0.007073
2003-2004	0.005898	0.007637	**0.002653**	0.003421	0.002725
2004-2005	0.006470	0.006107	**0.002867**	0.003966	0.003668
2005-2006	0.004693	0.006181	**0.003081**	0.003736	0.003474
2006-2007	0.003923	0.006061	0.004354	0.004146	**0.003134**
2007-2008	0.012549	0.009672	0.010819	0.015471	**0.008874**
2008-2009	0.014378	**0.006122**	0.013072	0.013401	0.008955

Table 5.3: Comparison of d_{32} for all models depending on the time interval.

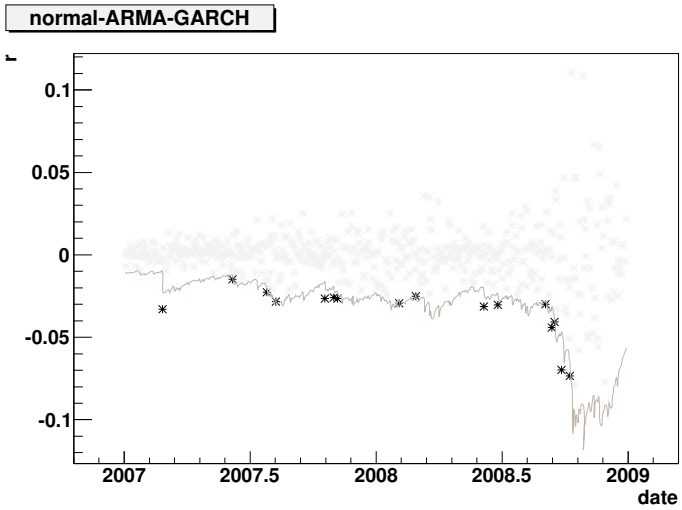

Fig. 5.14: Value at risk for the normal-ARMA-GARCH model.

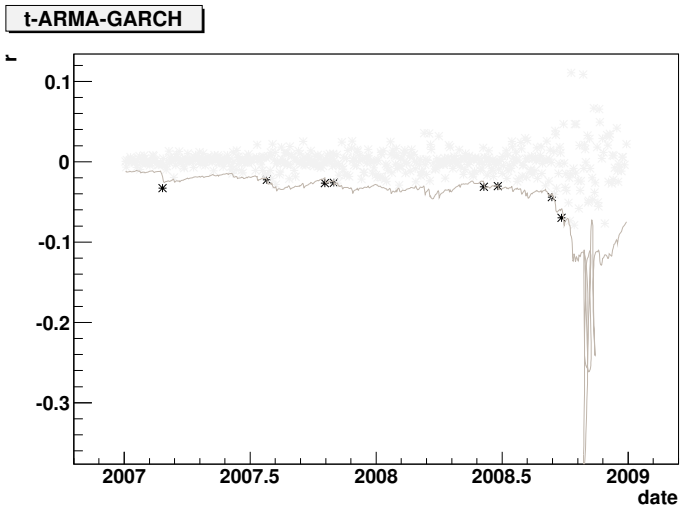

Fig. 5.15: Value at risk for the t-ARMA-GARCH model.

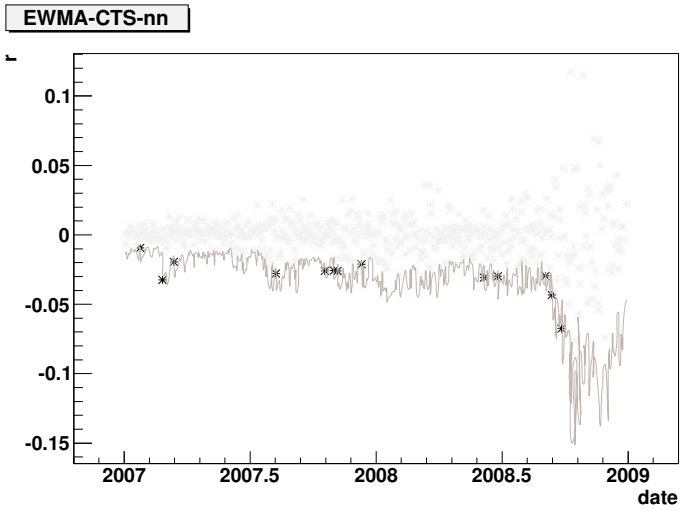

Fig. 5.16: Value at risk for the EWMA-CTS-nn model.

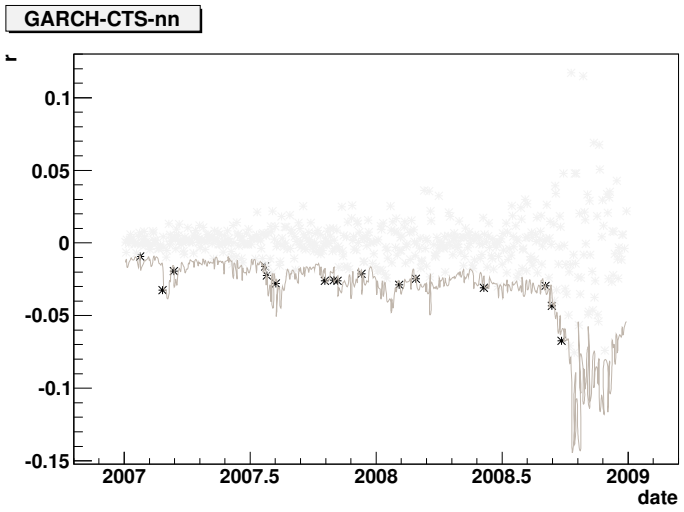

Fig. 5.17: Value at risk for the GARCH-CTS-nn model.

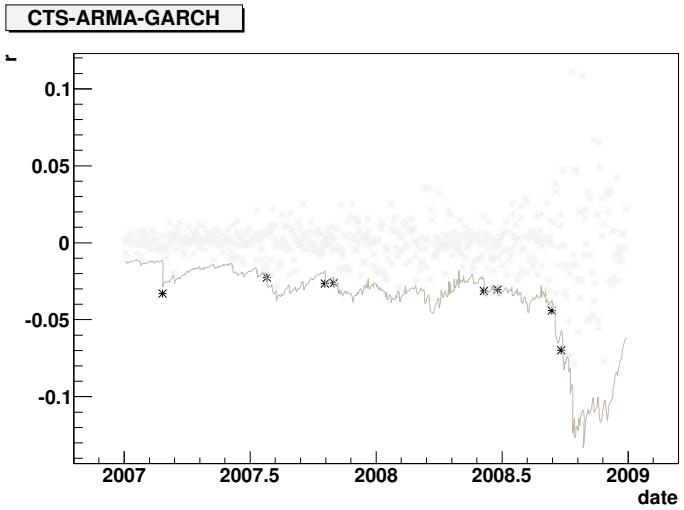

Fig. 5.18: Value at risk for the CTS-ARMA-GARCH model.

	normal-ARMA-GARCH	t-ARMA-GARCH	EWMA-CTS-nn	GARCH-CTS-nn	CTS-ARMA-GARCH
2007	7	**4**	8	10	**4**
2008	9	**4**	**5**	**6**	**4**

Table 5.4: Comparison of the value at risk violations in the years 2007 and 2008.

6

Conclusion

Forecasting time series is one of the most important tasks in finance. One of the most common approaches for forecasting is the application of GARCH models in which one has to explicitly assume a probability density function for the residuals. This is not necessary when employing neural network models since the network can learn the probability density function based on historical data. In order to make sure that modeling is within the APT framework, we fitted a classical tempered stable distribution to the output of the neural network.

In a large backtest with a time span of more than 20 years, our results show that our neural network model is able to forecast the time series of the DJIA with amazing precision and that it can outperform a t-ARMA-GARCH and a normal-ARMA-GARCH model. Our findings suggest that the neural network produces a similar prediction in the central area and in the tails, and the results are comparable to the results of a CTS-ARMA-GARCH model.

The forecasting abilities of a model is dependent on the time horizon which is used for the backtest. There is no universal model which performs well over every time period. When forecasting the risk of an asset one should use different advanced neural network models as well as advanced ARMA-GARCH models (such as CTS-ARMA-GARCH).

Part II

The Individualized Linear Regression

7

Introduction

One of the most common models in dealing with statistical problems is the linear regression analysis where typically one variable t_i depends linearly on x_i:

$$t_i = mx_i + b + \epsilon_i \tag{7.1}$$

and ϵ_i denotes the residual of event i. The parameters m and b have to be estimated. In the early 1960s, a good deal of research focused on analysis of the distribution of the residuals ϵ_t (see e.g. Anscombe and Tukey (1963)). Although the introduction of further regression variables decreases the variance of the residuals, using too many parameters usually leads to overfitting. As a result, stepwise procedures for estimating the amount of significant parameters were formulated, see Freund *et al.* (1961) and Goldberger (1961). One possibility in selecting variables for inclusion in the regression is the stepwise addition (Pope and Webster (1972)) or deletion (Mantel (1970)) of variables. The problems arising from these methods are reviewed in Hocking *et al.* (1976). A solution to these problems can be found by testing all combinations of variables which can result in a computational problem.

In the 1970s, people became aware that the assumption of variance homogeneity for the residuals, needed for the linear regression, is not fulfilled in many practical applications. If unknown parameters are fitted to a dataset, the procedure should react in a graceful way to minor deviations from the assumptions. Therefore many robust estimators were investigated, see e.g. Huber (1977). A further improvement of the linear regression was the development of a regime shifting multivariate linear regression. The basic idea is to define different regions in which the parameters are estimated. In Bai (1999) a likelihood ratio test is introduced, giving the number and locations of change points. Bai and Perron

(1998) describe how to successively estimate a break point in a multi-variate linear regression. Generalizations of the linear regression are consequently the nonlinear regression (e.g. Marquardt (1963), Hartley (1961)) and the regime shifting nonlinear regression (Johansen and Foss (1995)).

This paper presents an algorithm which does a linear regression in different regimes using a n-dimensional vector of exogenous variables. The regimes are not defined in a multi-dimensional space using all exogenous variables. Instead, we analyse the n projections with respect to the exogenous variables. The regimes are defined in each of the projections. In order to combine the n estimators of slope and intercept coming from the exogenous variables, we use mean and covariance of the estimators. We refer to the algorithm as an "individualized semi-linear regression". The parameters of the regression (slope and intercept) are assumed to vary continuously across the regimes. An important point is that we make use of spline fits (see Wahba (1990)), which makes the algorithm robust with respect to statistical outliers in the different regimes. Only statistically significant correlations of exogenous variables to the parameters of the regime shifted linear regression are identified. Linear correlations between exogenous variables are estimated and used to obtain good estimators for slope and intercept of an event. If the exogenous variables are not significantly correlated, the results of the individualized semi-linear regression converge to the results of the simple linear regression.

One example for the use of the algorithm is presented in a market impact analysis, see Fraenkle *et al.* (2010). A second example may be the analysis of the expected excess asset return to the expected excess market return in the capital asset pricing model (CAPM). In this application, the return of the asset is proportional to the return of the market. The slope (often refered to as β) depends on time. Therefore the time itself could be one of the exogenous variables. It may be analysed if β also depends on further exogenous variables such as market capitalization of the asset, traded volume, etc.

In section 8 we will describe how the algorithm works while section 9 will present different Monte Carlo simulations in which we compare the results of the individualized regression and a simple linear regression. We will conclude in section 10.

8

Description of the model

The basic idea of the invidivualized regression algorithm[1] originates from a regime shifted regression analysis (see Quandt (1958)). This means, slope m and intercept b can depend on exogenous variables $x_1, ..., x_n$. Thus we have

$$t_i = m(x_{1,i}, ..., x_{n,i})z_i + b(x_{1,i}, ..., x_{n,i}) + \epsilon_i \tag{8.1}$$

where t is the dependent variable, z the regressor and the residuals are defined by ϵ.
The first possible interpretation is that our model is able to find a functional nonlinear mapping between the input vector $x_{1,i}, ..., x_{n,i}, z_i$ and the target t_i. This nonlinear mapping is forced to be linear in one variable z, meaning that the projection of t with respect to z has to be linear. A second interpretation is that of a linear regression model which does not have a constant estimator for slope and intercept but which is able to define both individually for various regimes[2]. They are allowed to depend in a nonlinear way on the variables $x_1, ..., x_n$ which may be even correlated. This individualization of the parameters is the reason for referring to the model as an "individualized semi-linear regression". The third view point is that of a Taylor expansion with respect to z (while higher-order terms are allowed for $x_1, ..., x_n$ but omitted in z). To outline the ideas we have chosen an easy example of a typical problem and show how the algorithm is applied to solve it.

[1] All presented algorithms are also implemented in the NeuroBayes® software, developed by Phi-T® Physics Information Technologies GmbH.

[2] By the use of spline fits we will fit the results of different regimes which will lead to an eventwise definition of slope and intercept as will be explained in the next subsection.

8.1 The basic idea of the model

Let m and b depend on one exogenous variable, i.e. $n = 1$. Then equation (8.1) simplifies to a modified equation (7.1)

$$t_i = m(x_{1,i})z_i + b(x_{1,i}) + \epsilon_i \tag{8.2}$$

where the regression parameters depend on the variable x_1. This may be a state variable, denoting e.g. individual regimes of regressed data clusters. In Figure 8.1 we give a very simple example for a regime shifted regression. If one does a simple linear regression, the result is the black line in the middle which describes the set of data rather poorly. There are clearly two different regimes each of the data have to be described separately. The upper regime belongs to $x_1 = 1$ and the lower regime to $x_1 = -1$. Therefore the simple linear regression can be improved by using the exogenous variable x_1. In this example, the best solution is the fit of a regression in both regimes separately resulting in two distinct grey lines. The classification of the regimes is very obvious here, but in general we may have more than two regimes which may even not be visibly seperated.

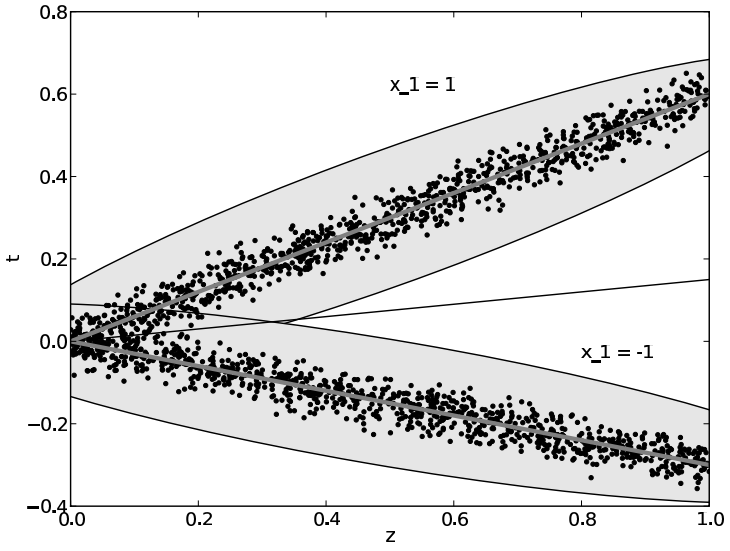

Fig. 8.1: Target t plotted dependent on variable z. The regression has to be done in two regimes depending on x_1.

One crucial idea in constructing the algorithm is the assumption of continuity: we as-

sume slope and intercept to vary slowly when changing the variable x_1. Next we can define subspaces in our data sample for similar values of x_1, and find an estimator of the slope and the intercept for this subspace. Generally, we divide the variable $x_{1,i}$ into k bins belonging to the k regimes. In order to have a comparable amount of statistics in each bin, we transform x_1 to a uniformly distributed variable by the probability integral transformation. In the subsequent step our parameters of the model are adjusted as follows: we estimate the slope $m_{1,j}$ and the intercept $b_{1,j}$ for every bin j of x_1 where $j \in [1, ..., k]$. Also to make the estimation of m_i more robust, we will use a spline fit to regularize statistical outliers. If we would like to have an estimator of m_i and b_i for event i, knowing x_1 and z, we have to look at the spline fit of slope and intercept of variable x_1 and get m_i^{pred} and b_i^{pred}. Then the prediction is $t_i = m_i^{\text{pred}} * z_i + b_i^{\text{pred}}$.
For one exogenous variable the method is easily understood and almost looks trivial. However, if we create a model with more variables, more enhanced methods will be needed.

8.2 The individualized semi-linear regression model

In the individualized regression we try to define slope and intercept of an event i which depends on exogenous variables $x_1, ..., x_n$. One main idea of the individualized regression is that if we want to estimate m_i and b_i for an event i, we analyse the n problems

$$\begin{aligned} t_i &= m(x_1)z_i + b(x_1) + \epsilon_i \qquad \Rightarrow m(x_1), b(x_1) \\ &\vdots \\ t_i &= m(x_n)z_i + b(x_n) + \epsilon_i \qquad \Rightarrow m(x_n), b(x_n) \end{aligned}$$

coming from the projections on the exogenous variables $x_1, ..., x_n$. We get estimators of m_i and b_i for all n exogenous variables. In the next step, we will combine these estimators appropriately to get one common estimator. The estimators for an event i have an additional index j denoting the estimator dependence of variable j ($j \in [1, ..., n]$). Thus we get the predictions

$$\Big(m_{i,1}\ m_{i,2}\ ...\ m_{i,n}\ b_{i,1}\ b_{i,2}\ ...\ b_{i,n}\Big)$$

One method to combine these particular predictions could be the computation of the mean for all m's and b's respectively. However this is not a good choice because it is possible that some variables have a large correlation to the target t_i whereas some other variables may not be correlated to the target at all. In this case one would like to give a larger weight

to the predicted values $m_{i,j}$ ($b_{i,j}$) coming from a variable j with a high correlation to the target.

A problem could also appear if the method is applied to a vector $\mathbf{x}$ in which all components are highly correlated to each other. The algorithm should recognize such correlations and make sure that the statistical significance of the correlation between the input variables and the target is not increased by introducing further redundant variables which would be highly correlated to the remaining variables. The desired algorithm should also deal with correlations among the input variables and be able to decide if a variable has a statistically significant correlation to m_i (b_i) at all. If there is a large correlation of a variable x_j to m_i (b_i), the weight of the estimator should be larger than the weight given to the estimator of an unimportant variable. Furthermore if the input variables are correlated among each other, the algorithm should treat these correlations correctly.

We assume the target t_i to be linear in the estimators $m_{i,1}z_i, ..., m_{i,n}z_i$ and $b_{i,1}, ..., b_{i,n}$. The n estimators of the slope (and the n estimators of the intercept respectively) originate from the n exogenous variables $x_{i,1}, ..., x_{i,n}$. We define $\mathbf{X}_i$ as

$$\mathbf{X_i} = \begin{pmatrix} t_i \\ m_{i,1}z_i \\ m_{i,2}z_i \\ \vdots \\ m_{i,n}z_i \\ b_{i,1} \\ b_{i,2} \\ \vdots \\ b_{i,n} \end{pmatrix} = \begin{pmatrix} t_i \\ \tilde{\mathbf{x}}_i \end{pmatrix} \tag{8.3}$$

where we summarize the estimators $m_{i,1}z_i, ..., m_{i,n}z_i$ and $b_{i,1}, ..., b_{i,n}$ in $\tilde{\mathbf{x}}_i$. If the target is linear in these estimators, we have to solve the multi-variate linear regression

$$t_i = a + \sum_{j=1}^{2n} w_j \tilde{x}_{i,j} + \epsilon_i \tag{8.4}$$

where a and w_j are parameters that have to be fitted. The result of this regression is given by (Chatfield and Collins (1980))

$$
\begin{aligned}
w_j &= (\Sigma_{t,\tilde{\mathbf{x}}}\Sigma^{-1}_{\tilde{\mathbf{x}},\tilde{\mathbf{x}}})_j \qquad j \in [1,...,2n] \\
a &= \mu_t - \mathbf{w}\boldsymbol{\mu}_{\tilde{\mathbf{x}}}
\end{aligned} \tag{8.5}
$$

where we introduced the mean $\boldsymbol{\mu}$ of $\mathbf{X}$

$$
\boldsymbol{\mu} = \begin{pmatrix} E(t) \\ E(m_1 z) \\ \vdots \\ E(m_n z) \\ E(b_1) \\ \vdots \\ E(b_n) \end{pmatrix} = \begin{pmatrix} \mu_t \\ \mu_{\tilde{\mathbf{x}}} \end{pmatrix} \tag{8.6}
$$

and the covariance matrix Σ of $\mathbf{X}$

$$
\Sigma = \begin{pmatrix} \Sigma_{t,t} & \Sigma_{t,\tilde{\mathbf{x}}} \\ \Sigma_{t,\tilde{\mathbf{x}}} & \Sigma_{\tilde{\mathbf{x}},\tilde{\mathbf{x}}} \end{pmatrix} \tag{8.7}
$$

Mean and covariance matrix have to be estimated from the data sample[3]. The target of an event can be predicted using (8.4) and the definition of the parameters in (8.5). So far we have explained how to get a prediction of the target. If we need the individualized slope and intercept for one event i, we can use the fact that the target is linear in $m_{i,j}z_i$ and in $b_{i,j}$ where $j \in [1,...,n]$. Basically the sum in (8.4) can be splitted into two sums. One sum includes the terms proportional to z_i. The second sum includes the predictions of the intercepts and the parameter a defined in (8.5). If the mean of z_i is zero, slope and intercept are not linearly correlated and we can identify the individualized slope with the sum of the weighted slopes and the individualized intercept with the rest. The results are

$$
\begin{aligned}
b_i^{\text{pred}} &= a + \sum_{j=1}^{n} w_{n+j} b_{i,j} \\
m_i^{\text{pred}} &= \sum_{j=1}^{n} w_j m_{i,j}
\end{aligned} \tag{8.8}
$$

[3] Therefore mean and covariance matrix have to exist. Since we use the χ^2-estimation for the linear regression, the mean has to be unbiased.

where a and $\mathbf{w}$ are defined in (8.5).
Here is a summary of all necessary steps:

- Transform z_i to have zero mean. Thus the z_i are stationary random variables.
- Transform all n input variables to a uniform distribution and define k bins in order to have approximately the same number of events in every bin.
- Select all events i for which variable x_1 is in the first bin.
- Fit a straight line with slope $m_{1,1}$ and intercept $b_{1,1}$ through the points (z_i, t_i) of the selected events using ordinary least squares.
- Continue with the rest of the bins of variable x_1.
- Plot $m_{1,j}$ $(j = 1, ..., k)$ as function of x_1 where x_1 is transformed to be uniformly distributed (and for b likewise)[4].
- Use a spline to fit m and b depending on x_1. An example for such a plot is shown in Figure 8.2.
- Proceed with variables $x_2, ..., x_n$ analogously.
- Compute mean $\boldsymbol{\mu}$ and covariance matrix Σ defined in (8.6) and (8.7) using $\mathbf{X}_i$ (defined in (8.3)) of all events.
- Compute the parameters a and $\mathbf{w}$ defined in equation (8.5) using mean and covariance matrix.
- Use equation (8.4) to get the conditional mean of the target for an event i.

In Figure 8.2 we give an example for a slope depending on variable x_1. The spline fits the slopes across the bins of x_1[5] smoothing statistical fluctuations.
The last bin in the spline fit has a special meaning. There are basically two different possibilities: It may happen that for some events i the variable x_j is unknown (or known to be incorrect). Then these events are separated into the last bin. The parameters m and b will then be estimated as for all other bins. The bin is not included in the spline fit because the meaning is completely different compared to the rest of the bins. The assumption that m and b are smoothly depending on the input variable is not valid for the last bin. It may also happen that there are no events in the training sample which are filled in the last bin. Then the estimator of m (b) is defined by the mean of all bins. If we want to predict an

[4] The algorithm is very robust under a variation of the number of bins. The reason is that the spline fit respects the error bars of slope and intercept. Therefore, if we increase k, the error bars increase and the results of the fit are still robust.
[5] In this example we have chosen 30 bins for x_1.

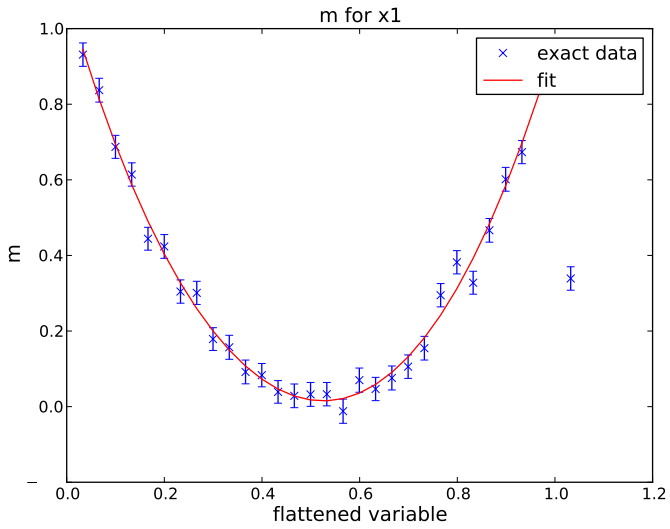

Fig. 8.2: Example of a fit of the slope m depending on variable x_1. We chose $k = 30$ bins in which we estimated m. The special meaning of the last bin is explained in the text. The error bars come from the linear regression and are used in the spline fit.

event in which the variable is not known (or wrong), the estimator of the special last bin is used.

9

Applications

In this section we will apply our algorithm to some different toy models and compare the results with a simple linear regression. For all examples we create a data sample of $N = 100000$ events. We use 50000 events to adjust the parameters of a simple linear regression and predict the 50000 remaining events (so we perform an out of sample test). The same is done for the individualized regression. The examples have the property that the target t_i is defined in equation (8.1) where ϵ follows a standard normal distribution. In order to compare the models we compute both the mean absolute deviation (mad) and the root mean square (for which we use the symbol σ). These are defined as follows:

$$mad = \frac{1}{N}\sum_{i=1}^{N} |m(x_{1,i}, ..., x_{n,i})z_i + b(x_{1,i}, ..., x_{n,i}) - t_i| \tag{9.1}$$

$$\sigma = \frac{1}{N}\sum_{i=1}^{N} (m(x_{1,i}, ..., x_{n,i})z_i + b(x_{1,i}, ..., x_{n,i}) - t_i)^2 \tag{9.2}$$

where t_i is the target of event i.

9.1 Example 1

In our first example we have 3 variables and slope and intercept are defined by

$$m_i = x_{1,i}^3 x_{2,i}^4 + x_{1,i} * x_{2,i} \qquad b_i = x_{1,i}^2 + x_{3,i}$$

The variables $x_1, ..., x_3$ are assumed to be uniformly distributed between -1 and 1 and the regression variable z follows a Gaussian distribution. As described in the last section we transform $x_1, ..., x_3$ to be uniformly distributed, select the t_i and z_i in the specific bins of

model	mad	σ
inclusive regression	1.00	1.55
individualized regression	0.84	1.12

Table 9.1: Comparison of the simple and the individualized semi-linear regression of example 1.

the variables $x_1, ..., x_3$ and estimate slope and intercept. The results can be seen in Figure 9.1. The fact that the slope depends nonlinear on x_1 and x_2 is well described by the spline fit. The slope does not depend on the variable x_3 which is also described correctly by the spline fit. The intercept only depends on x_1 and x_3. The results are presented in table 9.1 and show that the individualized semi-linear regression can outperform the simple linear regression. The fits of m and b are presented in 9.1. We did not include a stochastic component in the definition of m and b, so the model could easily learn the functional mappings that were used.

In order to verify that slope, intercept and the target variable t are predicted correctly, we present the profile plots (see Chapter B) in the Figures 9.2, 9.3 and 9.4. If the predictions of a random variable are correct, the profile plot should be compatible with the angle bisector[1].

[1] The reason is that the expectation value of a true value should be equal to the predicted value.

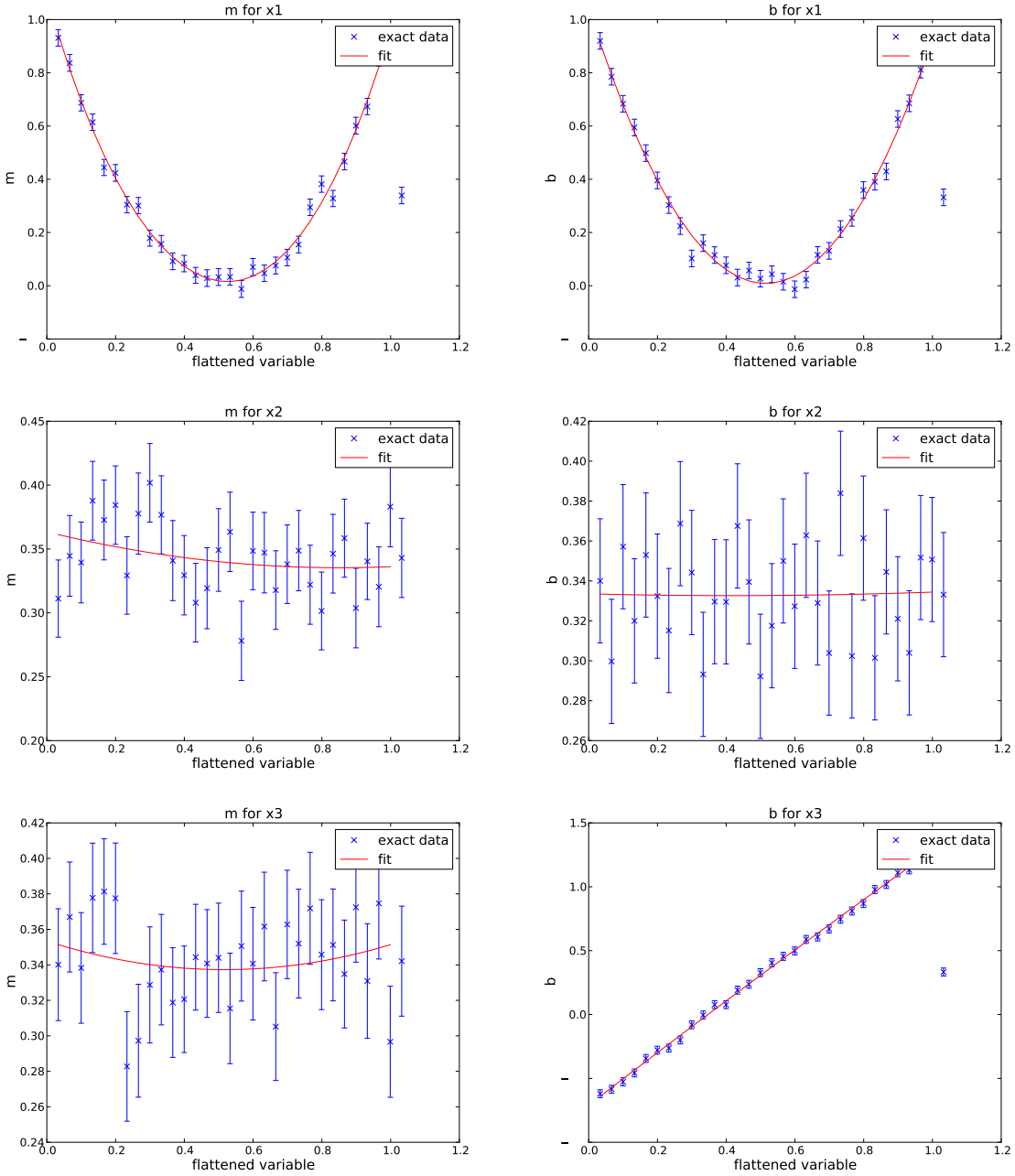

Fig. 9.1: Example 1: Dependency of slope and offset on the exogenous variables

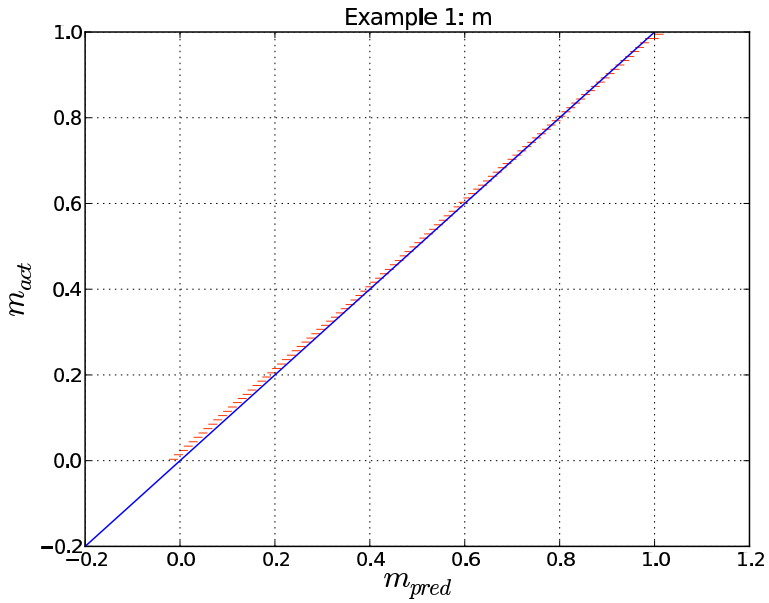

Fig. 9.2: Example 1: Prediction of the slope.

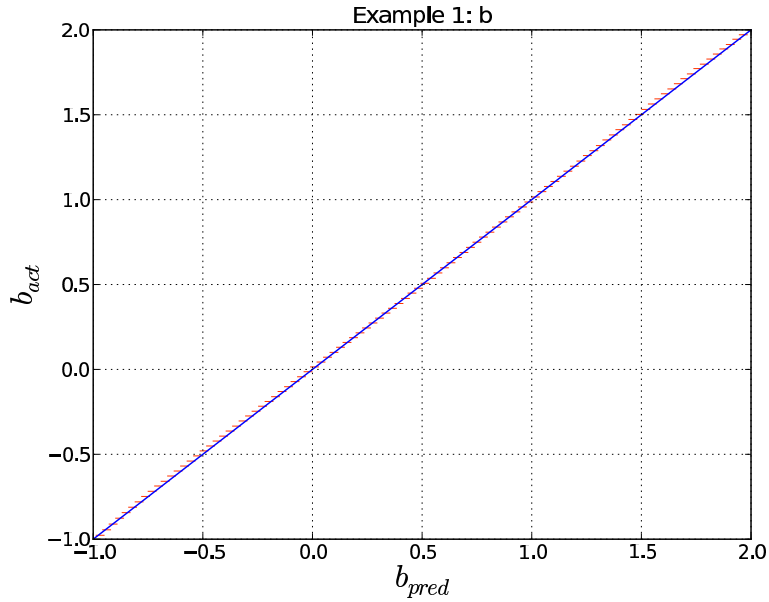

Fig. 9.3: Example 1: Prediction of the intercept.

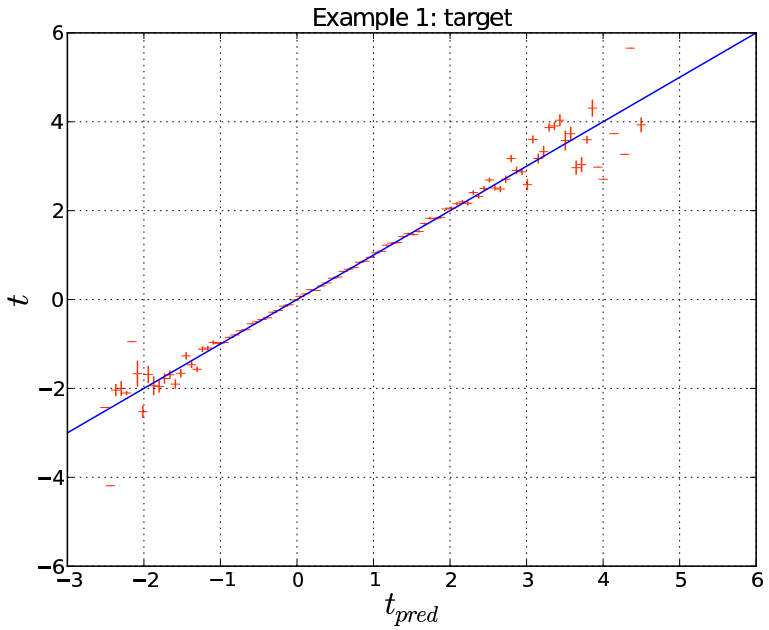

Fig. 9.4: Example 1: Prediction of the target.

9.2 Example 2

In the second example we will demonstrate that the algorithm also works if we introduce some stochastic components in the definitions of m and b. To increase the complexity of the problem we define a problem which has 6 independent variables. The variables $x_1, ..., x_6$ are uniformly distributed in the interval $[-1, 1]$ while the regression variable z is normally distributed. The functional mappings for the slope and the intercept are defined by

$$m_i = x_{1,i} + x_{2,i} + x_{3,i} * x_{4,i} + 0.5 * X \tag{9.3}$$

$$b_i = x_{5,i}^2 + x_{6,i} + Y \tag{9.4}$$

$$\tag{9.5}$$

where we introduced the stochastic variables X (which is normally distributed) and Y (which is uniformly distributed in the interval $[-0.5, 0.5]$). The results are summarized in table 9.2. Again the out of sample predictions of the individualized semi-linear regression perform better. The fits of the parameters m and b are able to describe the dependency from the input variables quite well (see Figures 9.5 and 9.6). The profile plots are shown in the Figures 9.7, 9.8, 9.9.

model	mad	σ
inclusive regression	1.04	1.74
individualized regression	0.92	1.35

Table 9.2: Comparison of the simple and the individualized semi-linear regression of example 2.

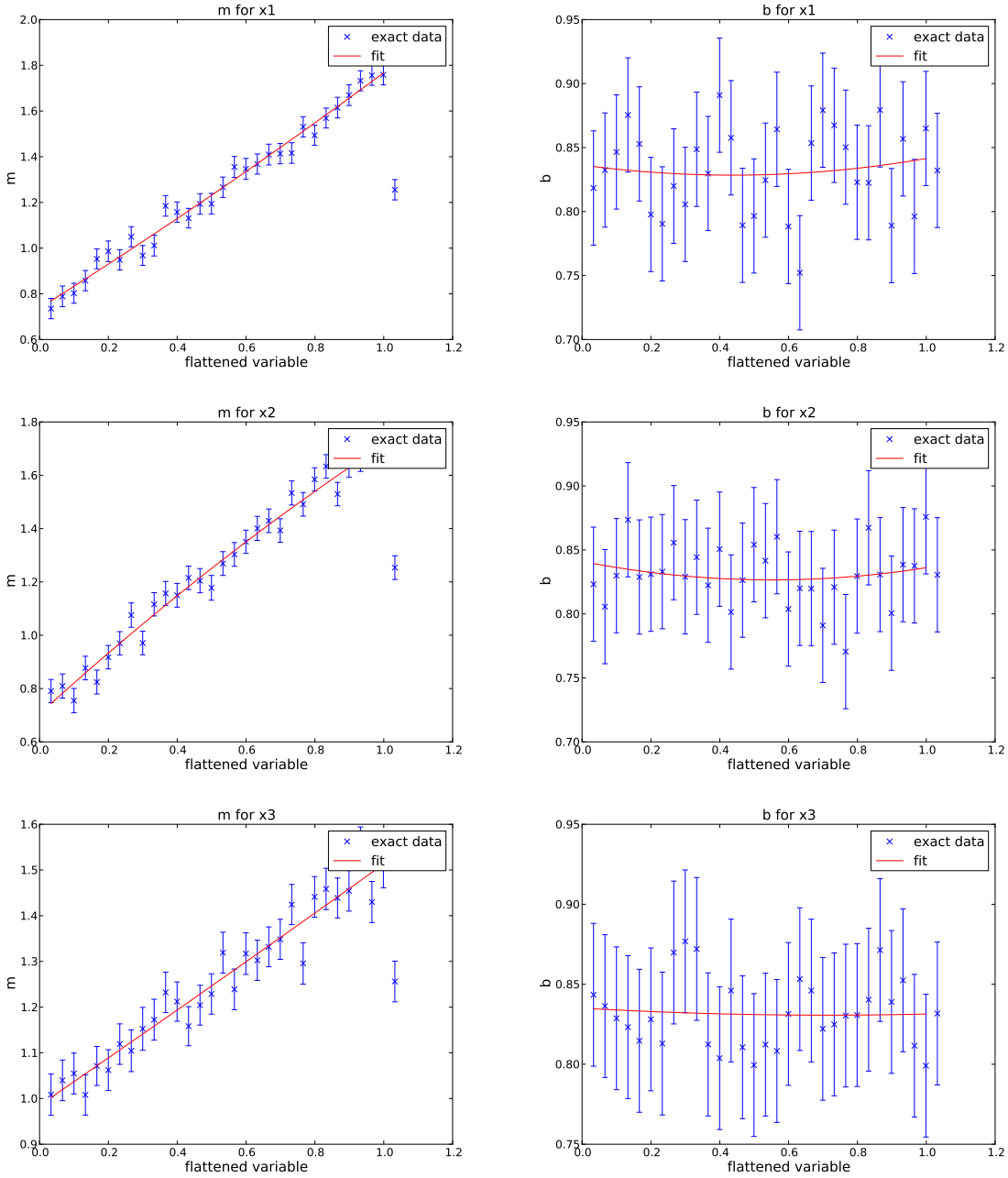

Fig. 9.5: Example 2: Dependency of slope and offset on the first three exogenous variables

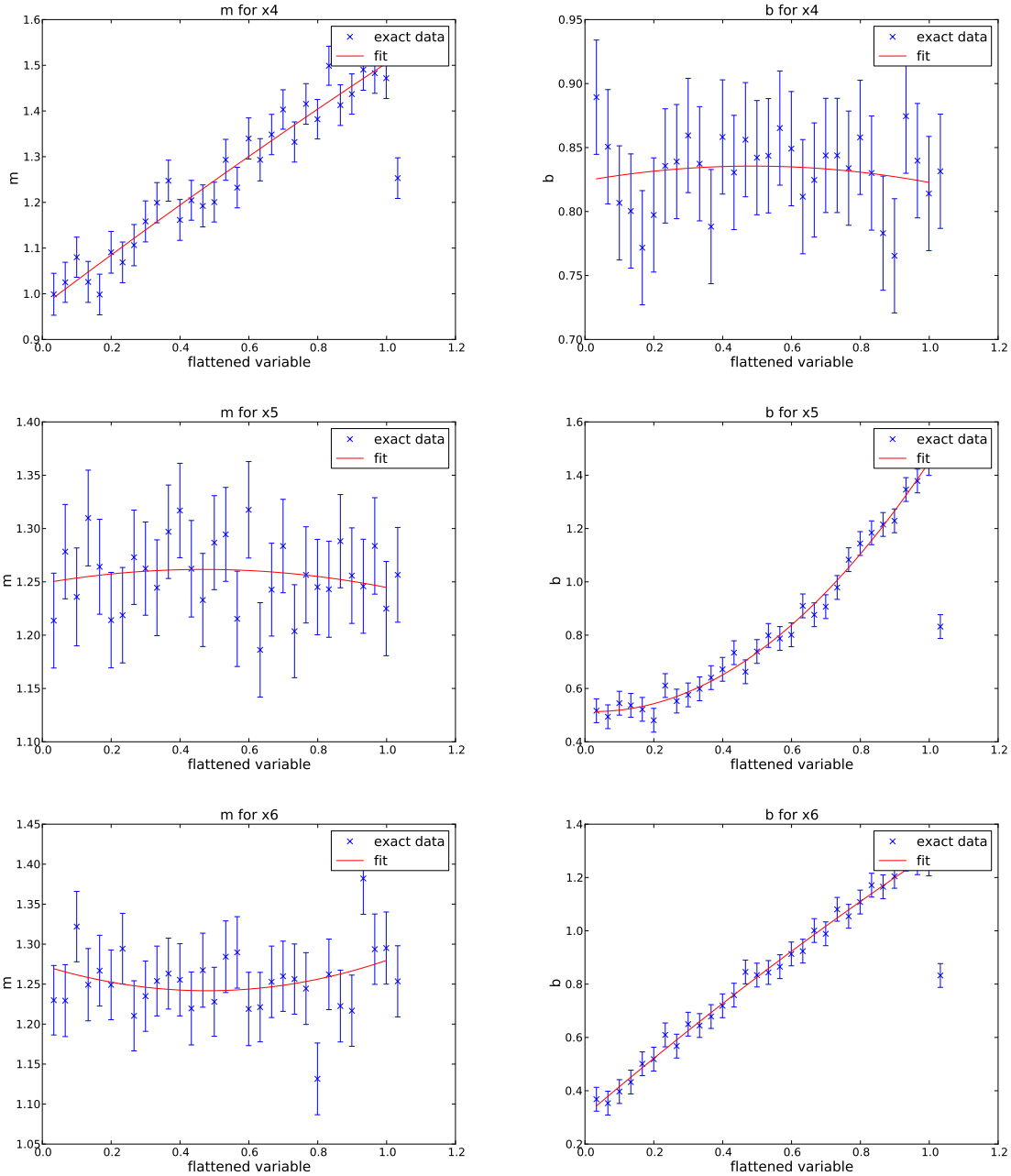

Fig. 9.6: Example 2: Dependency of slope and offset on the last three exogenous variables

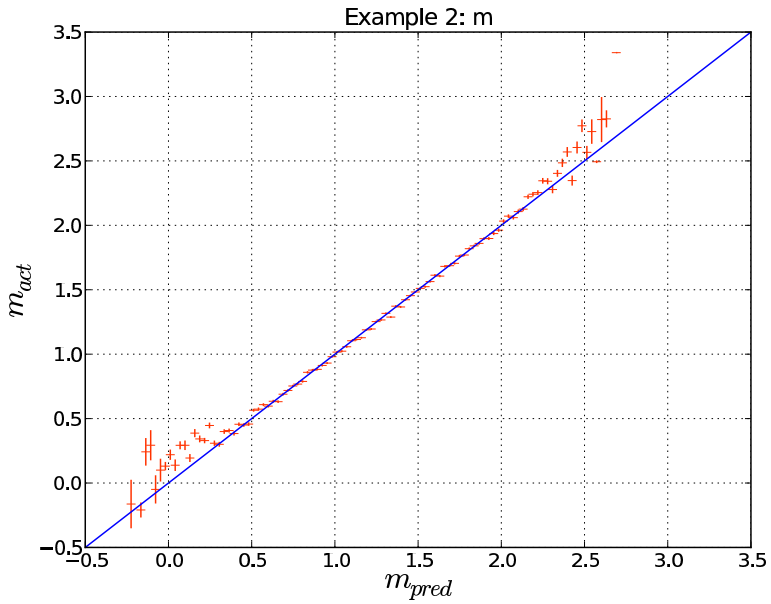

Fig. 9.7: Example 2: Prediction of the slope.

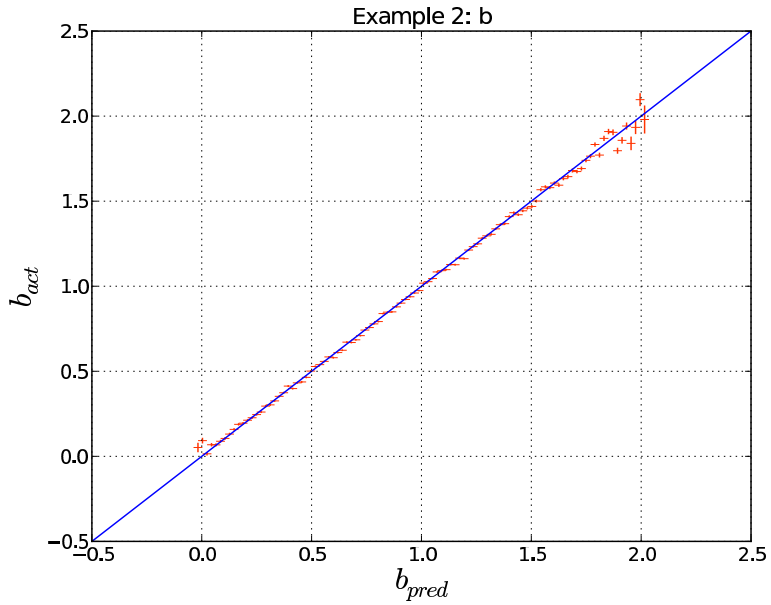

Fig. 9.8: Example 2: Prediction of the intercept.

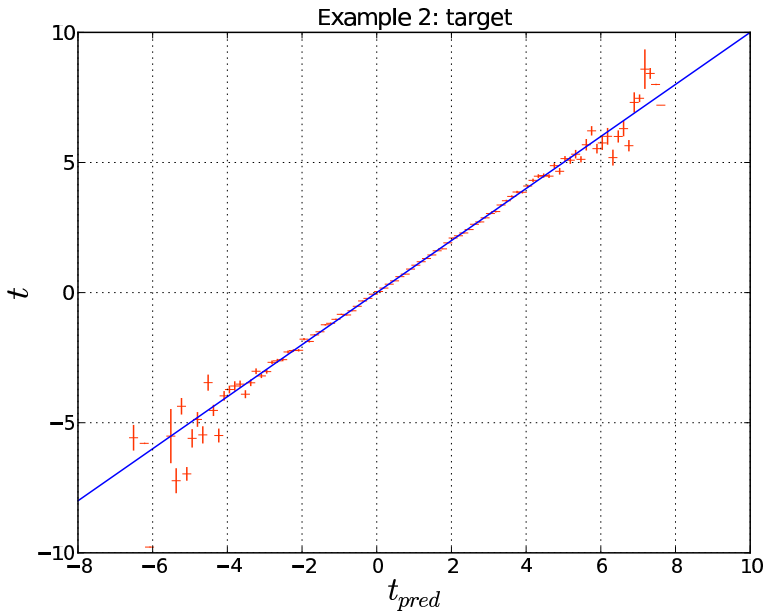

Fig. 9.9: Example 2: Prediction of the target.

9.3 Example 3

In this section we will show that the algorithm can deal with highly correlated exogenous variables. Therefore we start with the same definitions for slope and intercept which we have chosen in Section 9.2. Furthermore, we add three variables x_7, x_8, x_9 which are a sum of x_1 and a normal distributed random variable (X_7, X_8, X_9)

$$\begin{aligned} x_{7,i} &= x_{1,i} + X_7 \\ x_{8,i} &= x_{1,i} + X_8 \\ x_{9,i} &= x_{1,i} + X_9 \end{aligned} \tag{9.6}$$

The introduced three variables are nearly 100% correlated to the variable x_1. In table 9.3 we can see that the out-of-sample results for both semi-linear regression models are approximately equal. This illustrates the robustness of the algorithm with respect to correlated exogenous input-variables. The profile plots are illustrated in the Figures 9.10, 9.11, 9.12.

model	mad	σ
individualized regression (6 variables)	0.94	1.41
individualized regression (6 + 3 variables)	0.93	1.40

Table 9.3: Comparison of the two individualized semi-linear regressions.

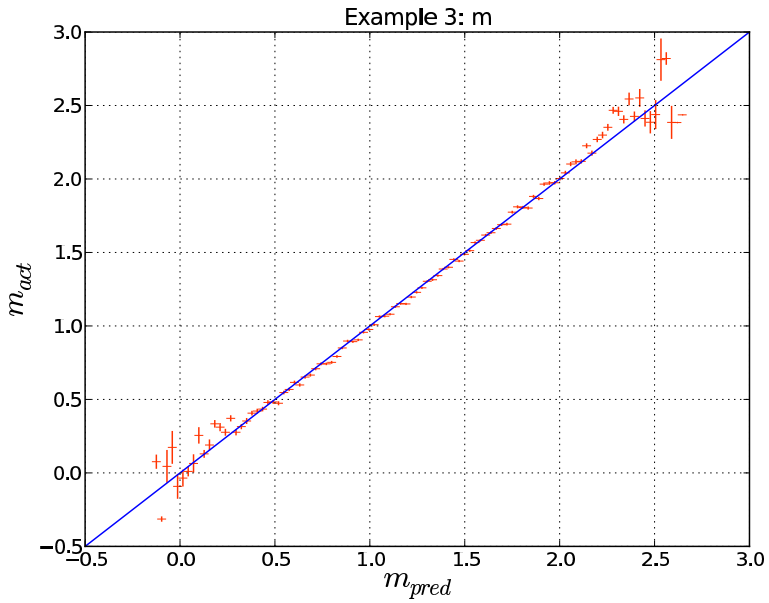

Fig. 9.10: Example 3: Prediction of the slope.

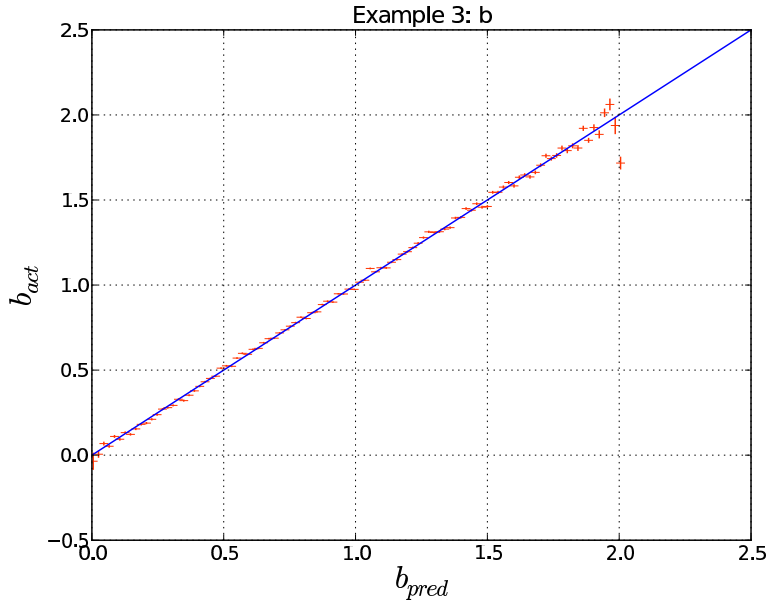

Fig. 9.11: Example 3: Prediction of the intercept.

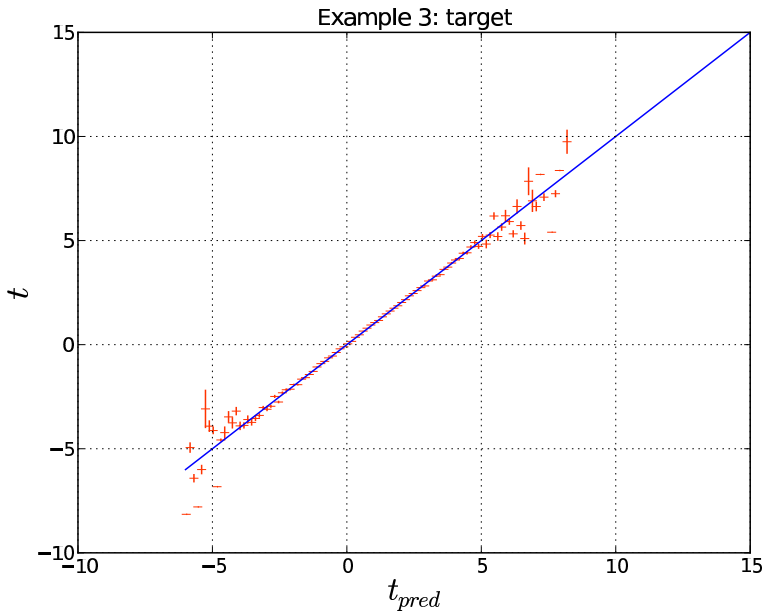

Fig. 9.12: Example 3: Prediction of the target.

9.4 Example 4

In our next example we use 3 input variables for the algorithm but the variables do not have a correlation to the slope and the intercept at all. In this case the algorithm should converge to the simple linear regression. This is a mere consistency check where we show that the algorithm is able to learn only statistically significant connections of the input variables and the parameters m and b. Here slope and intercept are given by

$$m_i = 0.5 * X + 1 \tag{9.7}$$

$$b_i = 0.5 * Y + 2 \tag{9.8}$$

$$\tag{9.9}$$

where X and Y are different standard normal distributed random numbers.
The fits are presented in Figure 9.4 and indeed there is no dependency of m and b from the input variables and the results (table 9.4) of both algorithms are consistent.

model	mad	σ
inclusive regression	0.97	1.50
individualized regression	0.97	1.50

Table 9.4: Comparison of the simple and the individualized semi-linear regression of example 4 in which there is no correlation between the parameters m and b to the input variables.

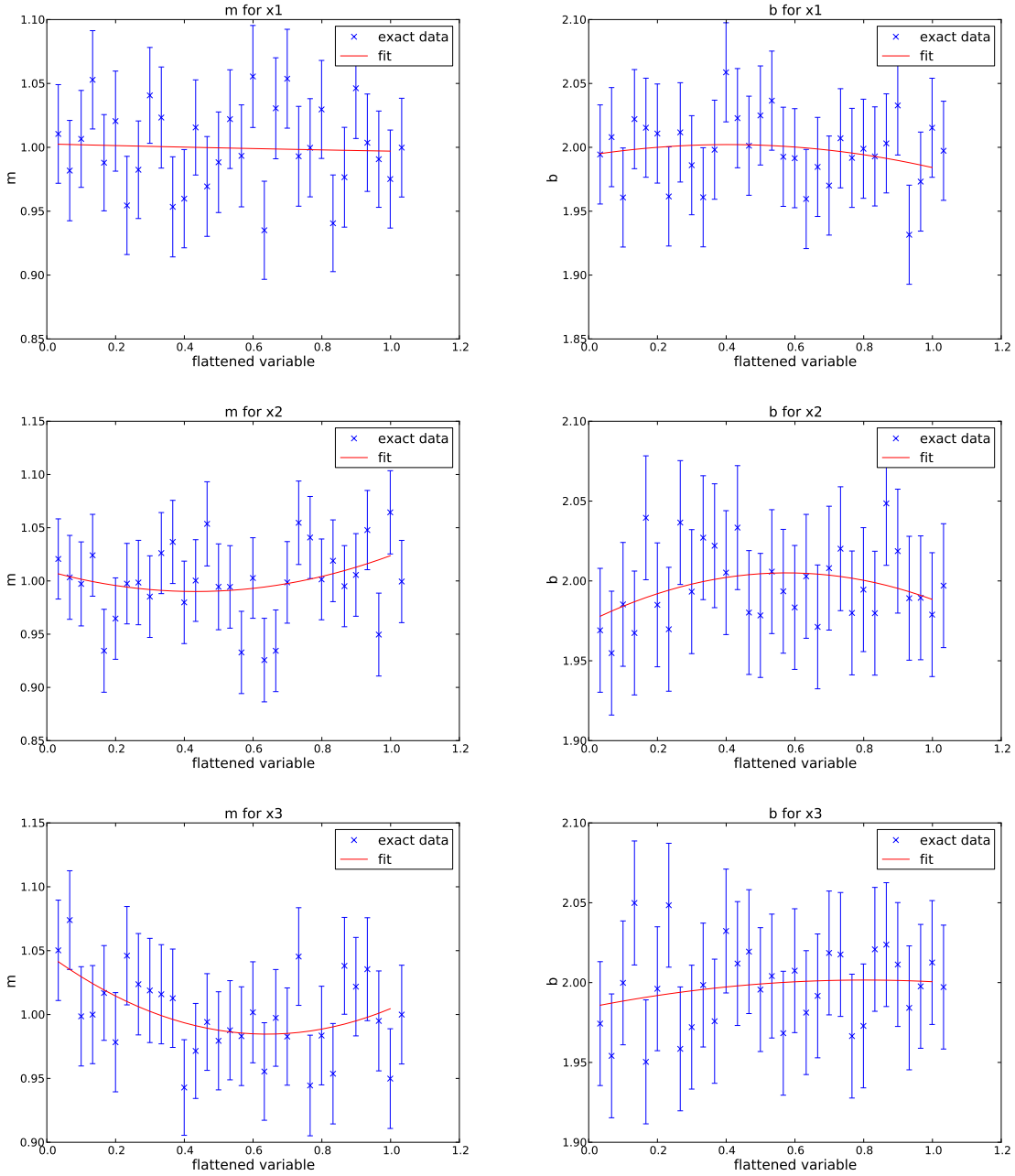

Fig. 9.13: Example 4: Dependency of slope and offset on the exogenous variables

10

Conclusion

The simple linear regression is one of the most common statistical tools. In this paper we presented a new algorithm which is basically an individualized semi-linear regression where slope and intercept are estimated for every single event. The algorithm is based on a regime shifted semi-linear regression using a continuous regime crossing. Therefore slope and intercept depend on an arbitrary amount of variables in a nonlinear way. The variables themselves can generally be correlated. Essentially we combined the ideas of regime shifting models, linear regression and nonlinear estimation to a powerful and extremely fast algorithm. It is robust with respect to outliers and works under generally fair assumptions (first and second moment of slope and offset have to exist). We have shown in several out of sample tests done with the help of Monte Carlo Simulations that the algorithm performs better than simple linear regression. If all used input variables for the algorithm are irrelevant for the regression, the result of the individualized regression converges to the result of the standard linear regression. Therefore the algorithm is consistent. The great success of this algorithm is to find a robust linear approximation with respect to one variable in a general multivariate nonlinear framework. Sometimes complex optimization problems are hardly solvable in a short time period, and it can be useful to get such a speedy linear approximation of an arbitrary nonlinear problem. In such a case the individualized linear regression can dramatically simplify the problem.

A

The Maximum Likelihood Method

In this chapter we examine some general concepts of parameter estimation. A detailed description about the estimation of parameters can be found by Blobel and Lohrmann (1998) and Cowan (1998).

We consider a random variable x and its probability density function (p.d.f.) $f(x|\mathbf{a})$ where $\mathbf{a} \in \mathbb{R}^n$. Also, we assume the functional form of $f(x|\mathbf{a})$ to be known but at least one component of $\mathbf{a}$ is unknown. The method of maximum likelihood can be used to estimate the values of these unknown parameters given a data sample $x_1, ..., x_n$. Under the assumption of the probability density $f(x|\mathbf{a})$, the probability of the i-th event in the data set in the interval $[x_i, x_i + dx_i]$ is obviously $f(x_i|\mathbf{a})dx_i$. Since the events are assumed to be independent, the probability 'x_i is in $[x_i, x_i + dx_i]$ for all i' is

$$P(\forall i : x_i \in [x_i + dx_i]) = \prod_{i=1}^{n} f(x_i|\mathbf{a})dx_i \tag{A.1}$$

If the hypothesized p.d.f. is correct, one expects a high probability for the events of the actually measured data set. Vice versa, a parameter value far away from the true value should yield a low probability for the measured events of the data set. Since dx_i does not depend on the parameters, the same argumentation also applies to the function $L(\mathbf{a})$

$$L(\mathbf{a}) = \prod_{i=1}^{n} f(x_i|\mathbf{a}) \tag{A.2}$$

This is called *likelihood function*. With this motivation one defines the maximum likelihood method which states that the best estimator $\hat{\mathbf{a}}$ of parameter $\mathbf{a}$ is the one maximizing the

likelihood function $L(\mathbf{a})$.

The maximum likelihood method turns out to be consistent, i.e. the estimated parameters converge in probability to the actual parameter $\mathbf{a}$ (see Blobel and Lohrmann (1998)):

$$\lim_{n\to\infty} P(|\mathbf{a} - \hat{\mathbf{a}}| > \epsilon) = 0 \tag{A.3}$$

Note that convergence in probability does not necessarily imply convergence in mean. Therefore, the described method is known to be not generally unbiased, i.e. the equation

$$E(\hat{\mathbf{a}}) = \mathbf{a} \tag{A.4}$$

is not necessarily true (see again Blobel and Lohrmann (1998)).

In practical applications it may be numerically difficult to compute the product of small likelihoods. For this reason it is common to maximize the logarithm of the likelihood $l(\mathbf{a})$:

$$l(\mathbf{a}) = \log(L(\mathbf{a})) = \log(\prod_{i=1}^{n} f(x_i|\mathbf{a})) = \sum_{i=1}^{n} \log(f(x_i|\mathbf{a})) \tag{A.5}$$

This is equivalent to the maximization of the likelihood because the logarithm is a monotonous function. Thus the maximum is invariant under this transformation.

B

Profile plot

A profile plot is a graphical data analysis technique in order to examine the relative behavior of a variable in a multivariate data set of length n. First we define two different random variables x and y. The most common graphical illustration of two variables is a scatter plot. In a scatter plot we plot (x_i, y_i), where $i \in [1, \ldots n]$. In order to illustrate this technique we assume the variables x and y to be distributed as follows:

$$x = N(0,1) \qquad y = N(0,1) + 0.1 \times x \tag{B.1}$$

where $N(0,1)$ is the normal distribution. Thus, both random variables are slightly correlated. The scatter-plot of x and y is shown in Figure B.1. In this scatter plot, we can hardly

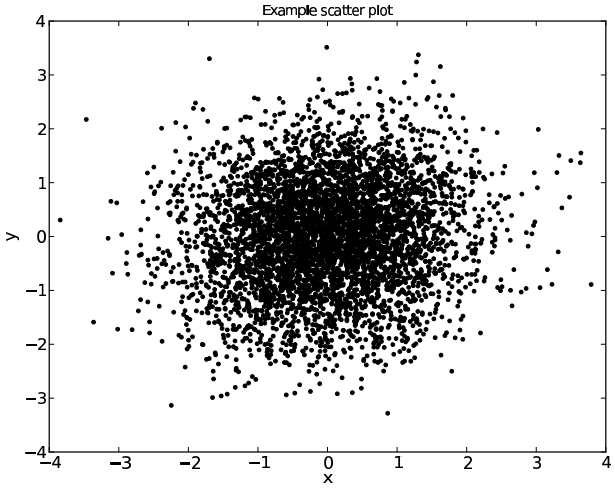

Fig. B.1: Example of a scatter plot

see the correlation between x and y. To this end we introduce the idea of the profile plot.

Profile plots are used to display the mean value of a variable y and its error for each bin in x. The mean of bin k is defined by

$$\mu_k = \frac{1}{m} \sum_{i=1}^{m} y_i^{(k)} \tag{B.2}$$

where the sum runs over all m events in which the variable x belongs to bin k. The error e_k of the mean μ_k is defined by (see Blobel and Lohrmann (1998))

$$e_k = \frac{\sigma_k}{\sqrt{m}} \qquad \sigma_k^2 = \frac{1}{m-1} \sum_{i=1}^{m} (y_i^{(k)} - \mu_k)^2 \tag{B.3}$$

The result is shown in Figure B.2. In the profile plot we can immediately see that variable y depends on variable x.

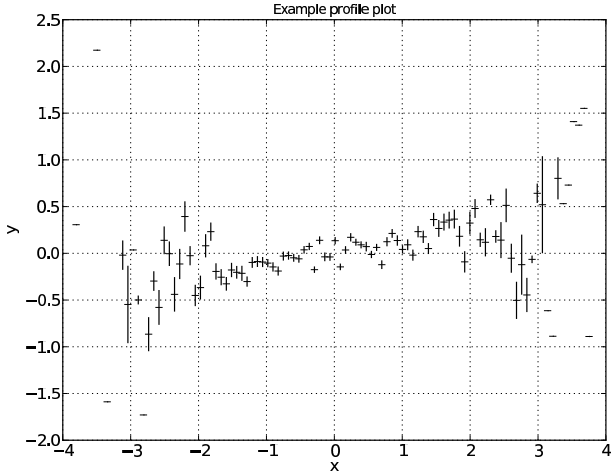

Fig. B.2: Example of a profile plot

C

Modeling Univariate Time Series

In this chapter we explain the mathematical background of discrete time series. We present the class of ARMA-GARCH models (see Bollerslev (1986b) and Engle (1982b)) which can be used to characterize and model observed time series. A detailed description of univariate as well as multivariate time series can be found by Rachev *et al.* (2007).
A discrete time series is a sequence $y_1, ..., y_T$ of realizations of a random variable y_t. We generally assume that all information which impacts the time series is included in the random variables. In order to model a time series mathematically, it is necessary that the process has certain properties, such as the *strict stationarity* which is defined by

$$F_{y_t, y_{t-1}, ..., y_{t-k}}(x_0, ..., x_k) = F_{y_{t-\tau}, y_{t-\tau-1}, ..., y_{t-\tau-k}}(x_0, ..., x_k) \tag{C.1}$$

where $F_{y_t, ..., y_{t-k}}(x_0, ..., x_k)$ is the joint cumulative distribution function of $(x_0, ..., x_k)$. Thus, stationarity means that the joint cumulative distribution function of k chronological events is invariant under translation. A weaker but in most cases adequate condition is the *weak stationarity* which is the invariance of mean μ and autocovariance $cov(y_t, y_{t-k}) = E[(y_t - \mu)(y_{t-k} - \mu)]$ under translation.

C.1 Autoregressive (AR) models

The basic idea of an auroregressive process (AR) of order p is that a value of a time series is a linear function of the past p values of the time series:

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + ...a_p y_{t-p} + \epsilon_t \tag{C.2}$$

In equation (C.2) the term ϵ_t is usually a 'small' random disturbance which is assumed to be independent and identically distributed (iid).
Using the lag operator L, with $Ly_t = y_{t-1}$, we can express (C.2) as

$$(1 - a_1 L - a_2 L^2 - ... - L^p a_p) y_t = a(L) y_t = \epsilon_t \tag{C.3}$$

where we introduced the autoregressive polynomial $a(L)$. The solution y_t of the so-called "difference equation" (C.3) will be connected to the roots of the autoregressive polynomial. Here, roots are defined by the solution of the inverse characteristic equation $a(\lambda) = 0$, with a complex variable λ. For weak stationarity of the time series it is necessary that all absolute values of its roots are larger than 1. Then the time series can be modeled with the help of an autoregressive process.

C.2 Moving Average (MA) models

A moving average process (MA) of order q depends on the last q disturbances:

$$y_t = b_1 \epsilon_{t-1} + ... + b_q \epsilon_{t-q} + \epsilon_t = b(L) \epsilon_t \tag{C.4}$$

The process y_t is stationary if the contition $\sum_q^{i=1} |b_i| < \infty$ holds.
The Wold decomposition theorem states that every weak stationary time series can be presented by a $MA(\infty)$ process, see Rachev *et al.* (2007).
Conversely, if a time series is *invertible*, i.e., the roots of the moving average polynomial are outside the unit circle, the $MA(q)$ model can be expressed by a $AR(\infty)$ model.

C.3 ARMA models

In an autoregressive moving average model (ARMA) we combine the $AR(p)$ process and the $MA(q)$ process to the $ARMA(p, q)$ process

$$y_t = \sum_{i=1}^{p} a_i y_{t-i} + \sum_{i=1}^{q} b_i \epsilon_{t-i} + \epsilon_t \tag{C.5}$$

where again the residuals ϵ_i have to be iid.
One can show that stationarity of an $ARMA(p, q)$ process depends reclusively on the parameters of the $AR(p)$ process (Rachev *et al.* (2007)).

C.4 GARCH models

The assumption that the residuals ϵ_t in equation (C.5) are iid is typically not valid in financial time series. The width of the residuals is clustered and depends on the time itself. This property is called volatility clustering. Engle and Bollerslev introduced the class of $GARCH(r, s)$ models which covers this property, see Bollerslev (1986b) and Engle (1982b). The volatility h_t can be modeled by expressing the residual term as follows:

$$\epsilon_t = h_t \eta_t \tag{C.6}$$

with the recursive relation

$$h_t^2 = \alpha_0 + \sum_{i=1}^{r} \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^{s} \beta_i h_{t-i}^2 \tag{C.7}$$

and η_t is iid. Thus, the volatility depends on its previous values and on squared residuals η_t^2. There are different possibilities how to get estimators of the initial values $h_0, h_1, \ldots$, see Rachev *et al.* (2007).

C.5 ARMA-GARCH models

While a GARCH model can describe volatility clustering, one still needs an adequate model to estimate the conditional mean of the time series. If the prediction of the conditional mean is not reasonable, the construction of the true conditional volatility is not possible.
Therefore, one combines the models discussed in this section to the class of $ARMA - GARCH(p, q, r, s)$ models which are defined by

$$y_t = \sum_{i=1}^{p} a_i y_{t-i} + \sum_{i=1}^{q} b_i \epsilon_{t-i} + \eta_t h_t \tag{C.8}$$

where h_t is given by equation (C.7) and η_t is assumed to have mean zero and variance one. For η_t we can assume various probability density functions to test different types of ARMA-GARCH models.
There are two approaches for the estimation of the parameters of an ARMA-GARCH process:

1. Estimate all parameters in one step using the Maximum Likelihood Method

2. Estimate the parameters of the ARMA model in the first step and the parameters of the GARCH model in the second step

The second method is often refered to as GARCH estimation after linear filtering.

C.6 Distributions for the innovations

This section presents some different distributions in order to model the innovations of an ARMA-GARCH process. After the introduction of the normal distribution and the t-distribution we cover the topic of α-stable and tempered stable distributions which include skewness and heavy tails, a property which can often be found in financial time series.

C.6.1 Normal distribution

The normal distribution is still the most common distribution used in finance. The pdf is defined by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{C.9}$$

where μ is the mean and σ is the standard deviation.
There are several reasons for the importance of the normal distribution in finance:

- By the central limit theorem the distribution of a sum of independent random variables which are arbitrary distributed and have finite first and second moments, converges to a normal distribution. Therefore, the normal distribution appears 'naturally' in many fields.
- There is an abundance of statistical tests for a process in which the normal distribution is the underlying pdf.
- For the normal distribution exists a risk-free process. Therefore one can compute prices for derivatives, such as options and futures, in the framework of arbitrage pricing theory.
- The parameters μ and σ can easily be estimated from a data sample.

Although the normal distribution has mathematically attractive properties, the usefulness in practical applications in finance is questionable as the returns of a financial time series are typically nonsymmetric and fat-tailed. Thus, they cannot be accurately described by a normal distribution. This was first noted by Mandelbrot (1963).

C.6.2 t-distribution

The t-distribution is defined by

$$f(x) = \frac{1}{\sqrt{n\pi}} \frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} \tag{C.10}$$

where n is the degree of freedom.
The t-distribution is able to describe fat tails. The distribution is symmetric, therefore it is not possible to fit an asymmetric distribution adequately.

C.6.3 Stable distributions

Motivated by Mandelbrot (1963) various non-normal distributions have been suggested in literature. One of the most promising class of distributions is the class of so-called stable distributions (see among others Rachev *et al.* (2005), Rachev *et al.* (2007), Rachev (2003)). Stability means that the distribution of the process does not depend on the scale, or more precisely

$$X_1 + ... + X_n \stackrel{d}{=} c_n X + d_n \tag{C.11}$$

where this equality refers to equality in distributions. Thus, the distribution function of a stable process is invariant under changes of the underlying time intervals up to scale and location.
The stability is a beautiful mathematical property and very important e.g. in risk management or portfolio analysis, where the probability density functions of different assets have to be combined. Using a non-stable distribution may result in numerical problems because Monte Carlo simulations have to be done in order to combine the probability density functions.

A second important property of the stable distributions results in the Central Limit Theorem: If one adds n iid variables having finite variance, the sum converges to a normal distribution. If the variance is infinite, the sum converges to a stable distribution. Therefore, the family of stable distributions is important in finance. If the price changes of an asset are driven by many iid shocks having infinite variance, the asset return can be described by a stable distribution.

C.6.3.1 α-stable distributions

A α-stable distribution is specified by four parameters. Furthermore, it can model asymmetry and fat tails which are important properties of a financial time series. Though it belongs to the class of stable distributions, there are still some disadvantages which complicate the use in mathematical financial models which we will discuss in this subsection.
The characteristic function, i.e., the Fourier transform of the probability density function, is defined as

$$E(e^{ikx}) = \begin{cases} \exp\left[i\mu k - \sigma|\mu|^\alpha \left(1 - i\beta \operatorname{sign}(k) \tan\left(\frac{\pi\alpha}{2}\right)\right)\right] & \text{if} \quad \alpha \neq 1 \\ \exp\left[i\mu k - \sigma|\mu| \left(1 + i\beta\frac{2}{\pi} \operatorname{sign}(k) \ln|k|\right)\right] & \text{if} \quad \alpha = 1 \end{cases}$$

where

$$\operatorname{sign}(k) = \begin{cases} 1 & \text{if} \quad k > 0 \\ 0 & \text{if} \quad k = 0 \\ -1 & \text{if} \quad k < 1 \end{cases}$$

The parameters μ, σ, α and β can be interpreted as follows:

- $\mu \in \mathbb{R}$ is the location parameter
- $\sigma > 0$ is the scale parameter
- $\alpha \in (0, 2]$ is the index of stability
- $\beta \in [-1, 1]$ is the skewness parameter

The tail behaviour is asymptotically $x^{-\alpha-1}$. Therefore, the moments $E(X^p)$ do not exist for $p \geq \alpha$. Furthermore, the variance does not exist, and the mean only exists for $\alpha > 1$. These properties have consequences for practical applications such as portfolio optimization, risk management, etc. For a portfolio one will expect an infinite gain and an infinite risk. Therefore decisions cannot be made using mean and variance.

C.6.3.2 Tempered stable distributions

Whereas the normal distribution does not permit to model fat tails and asymmetry, the class of α-stable distributions includes these properties. However, it is often a practical problem if the moments are infinite. The idea of modeling fat tails and skewness without having infinite moments can be combined to the class of tempered stable distributions.

The tempered stable distributions where originally introduced in statistical physics by Koponen (1995) under the name "truncated Levy flight". They were reconsidered in Barndorff-Nielsen and Shephard (2001) and Cont and Tankov (2004) as "tempered stable distribution", as "KoBoL distribution" by Boyarchenko and Levendorskiĭ (2002) and used as "CGMY" by Carr *et al.* (2002). Rosinski (2007b) generalized the classical tempered stable distribution (CTS) referring to it as the "tempered stable distribution". He shows that a tempered stable distribution can be parameterized similar to a stable distribution and that a tempered stable stochastic process looks like a stable process on small time scales and like a Brownian motion in long time scales.

A CTS distribution is defined as follows:

> *A random variable X is said to follow the classical tempered stable distribution if its characteristic function is given by*
>
> $$\begin{aligned}\Phi_X(u;\alpha,C,\lambda_+,\lambda_-,m) = \\ \exp(ium + C\Gamma(-\alpha)((\lambda_+ - iu)^\alpha - \lambda_+^\alpha) \\ + C\Gamma(-\alpha)((\lambda_- + iu)^\alpha - \lambda_-^\alpha))\end{aligned}$$
>
> *where $C, \lambda_+, \lambda_- > 0$, $\alpha \in (0,2)$ and $m \in \mathbb{R}$. A Lévy process induced from the CTS distribution is called a classical tempered stable process with parameters $(\alpha, C, \lambda_+, \lambda_-, m)$.*

If

$$C = (\Gamma(2-\alpha)(\lambda_+^{\alpha-2} + \lambda_-^{\alpha-2}))^{-1} \tag{C.12}$$

$$m = -\Gamma(1-\alpha)(C\lambda_+^{\alpha-1} - C\lambda_-^{\alpha-1}) \tag{C.13}$$

is fulfilled then $X \sim \text{CTS}(\alpha, C, \lambda_+, \lambda_-, m)$ has zero mean and unit variance and we call X the *standard CTS distribution* denoted by $\text{stdCTS}(\tilde{\alpha}, \tilde{\lambda}_+, \tilde{\lambda}_-)$ where we use the tilde to symbolize the parameters of the standard CTS distibution.

If we use the definition of the characteristic function (C.12) we can define the cumulants $c_n(X) \equiv \frac{1}{i^n}\frac{d^n}{du^n}\log(E[e^{iuX}])$:

$$c_n(X) = \begin{cases} m + \Gamma(1-\alpha)(C\lambda_+^{\alpha-1} - C\lambda_-^{\alpha-1}), & \text{for } n = 1 \\ \Gamma(n-\alpha)(C\lambda_+^{\alpha-n} + (-1)^n C\lambda_-^{\alpha-n}), & \text{for } n = 2, 3, ... \end{cases} \tag{C.14}$$

The parameters m and C determine location and scale of the distribution. The dependency of C is illustrated in Figure C.1. In Figure C.2 we show the effect of the parameter α. The smaller the parameter α, the larger becomes the peak in the center of the distribution. The parameters λ_+ and λ_- characterize the asymmetry. In Figure C.3 we can see the effect of a variation of the parameter λ_+.

Furthermore, increasing λ_+ and λ_- simultaneously (see C.4 for the CTS and Figure C.5 for the standard-CTS) results in a faster than α-stable decay. In order to illustrate this fact, we plot the logarithm of the standard-CTS-distribution in Figure C.6 where we increase λ_+ and λ_- simultaneously.

For $\alpha \to 2$ the CTS distribution converges to a Gaussian. This effect is shown in C.7 where we plot the logarithm of the standard-CTS. The central area looks like a Gaussian while the beginning of the tails depends on λ_+ and λ_-. For small λ_+ and λ_-, the non-Gaussian tails begin near the center. For a large λ_+ and λ_-, the non-Gaussian tails begin far away from the center, see C.8.

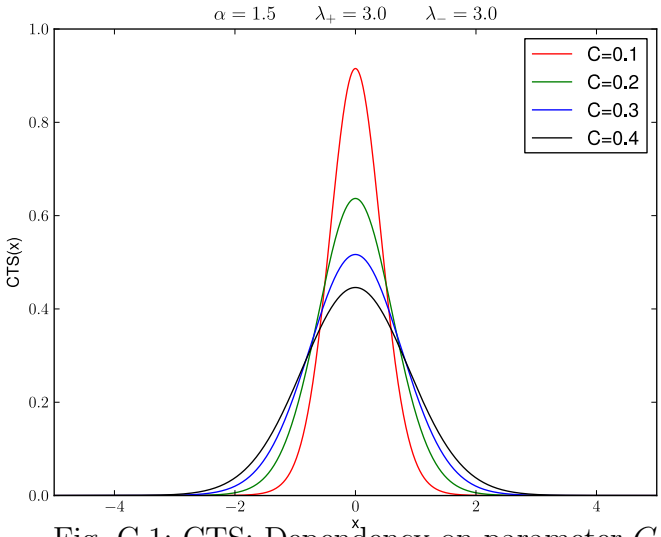

Fig. C.1: CTS: Dependency on parameter C

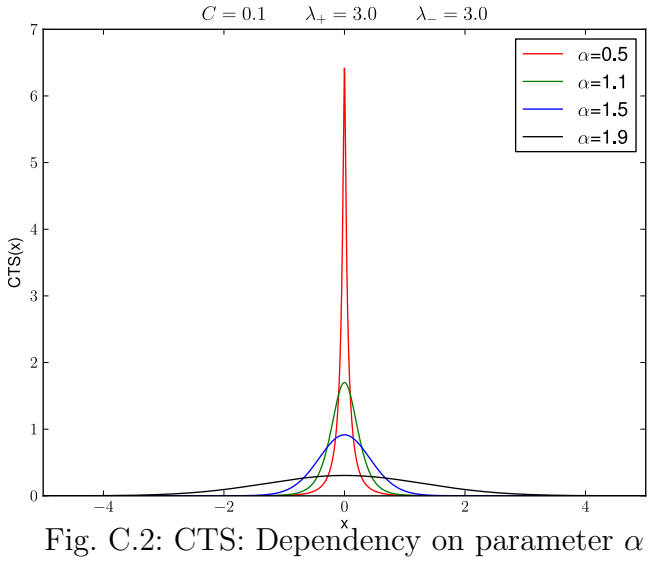

Fig. C.2: CTS: Dependency on parameter α

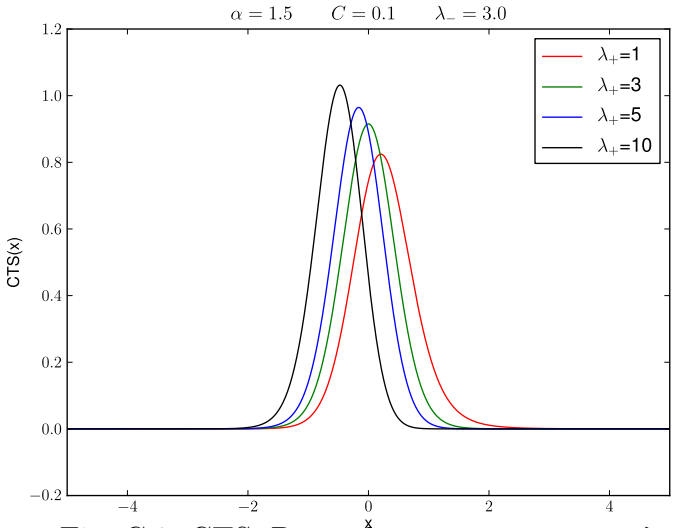

Fig. C.3: CTS: Dependency on parameter λ_+

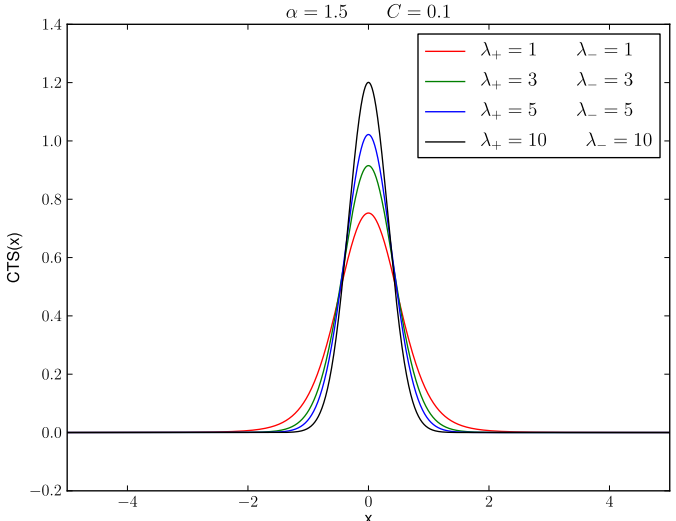

Fig. C.4: CTS: Varying λ_+ and λ_- simultaneously

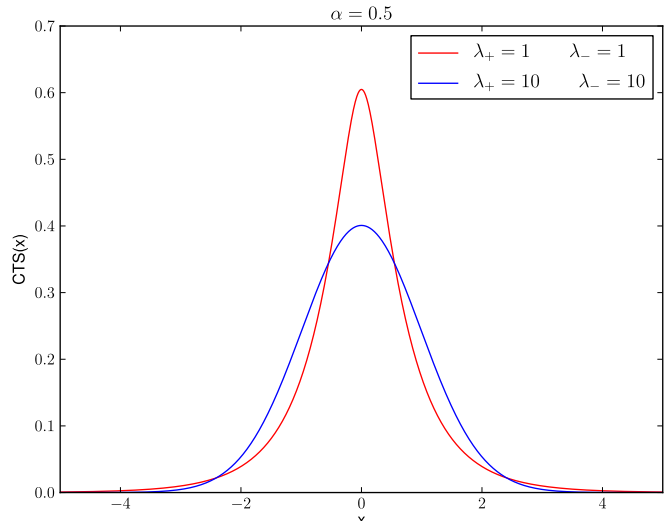

Fig. C.5: stdCTS: Varying λ_+ and λ_- simultaneously

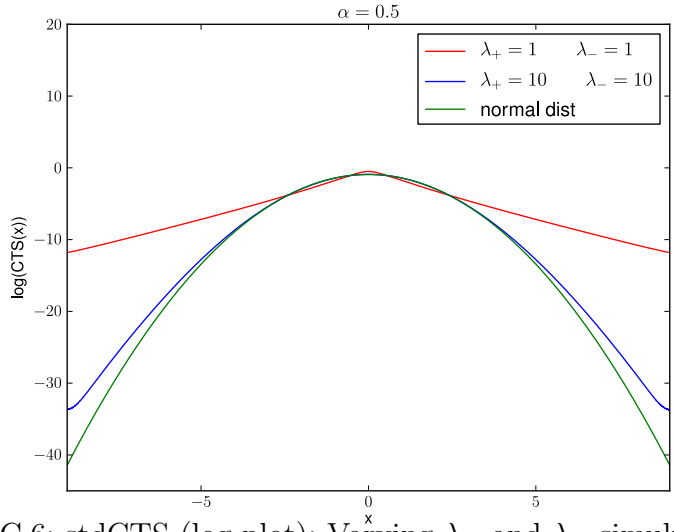

Fig. C.6: stdCTS (log-plot): Varying λ_+ and λ_- simultaneously

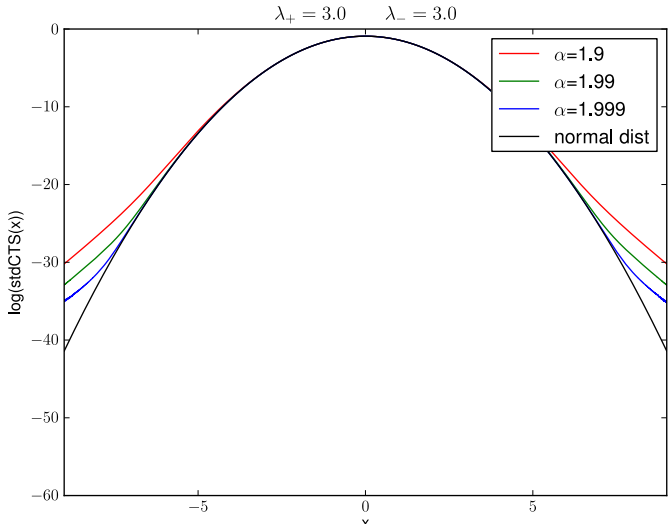

Fig. C.7: stdCTS (log-plot): Varying α

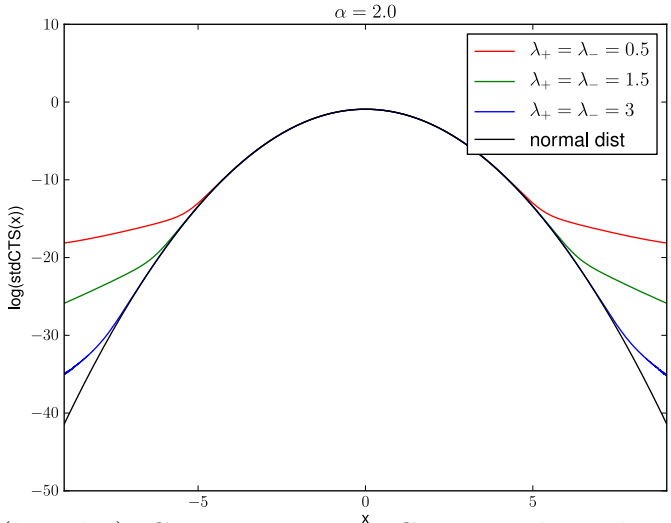

Fig. C.8: stdCTS (log-plot): Convergence to a Gaussian dependent on the parameters λ_+ and λ_-.

D

Neural networks

Neural networks have seen an explosion of interest in the last decades. They are successfully applied in many areas such as engineering, medicine, phsyics, geology and finance. In this chapter we summarize the basic ideas of neural networks and present the most important algorithms. A short introduction can be found in Haag (2003).
Originally, neural networks grew out of research in artificial intelligence where it was tried to model the low-level structure of a brain. One of the most important properties of a brain is the capability that it can reorganize itself and is able to *learn*, using simple but many processing units, the so-called neurons. A Neuron is a cell which has the ability to propagate an electrochemical signal. It consists of a branching input structure (the dentrites), a cell body and a branching output structure (the axon). When a neuron is activated, it sends an electrochemical signal along the axon. A neural network tries to model this structure mathematically. A further property of a brain is that knowledge, gained from historical events, can be applied in order to solve new similar problems. Therefore, a high degree of fault tolerance against noisy data is needed.

D.1 The basic mathematical concept

Using the idea of biological neural systems, one can construct a mathematical model of such a neuron. Each neuron receives input information from neurons of the neighborhood or external sources. The input information x_i is used to calculate an output y_i that is transfered to neurons or external sources. A technical neural network consists of many neurons which are connected by directed, weighted connections. A weight (or strength) of the connection between a neuron i and a neuron j is refered to as $w_{i,j}$. The propagation function of a neuron, which converts the input vector $\mathbf{x}$ to the output scalar $y(\mathbf{x})$, is then

defined by

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{D.1}$$

A monotonic activation function $F(y)$ has to be defined and does determine the activation

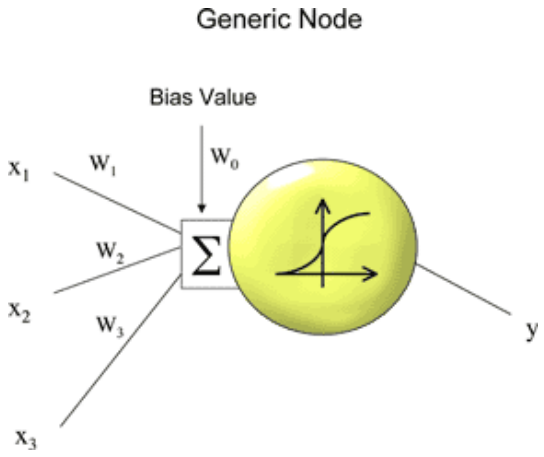

Fig. D.1: This is an illustration of a neuron. The sum represents the sum of the weighted inputs. The sigmoid function stands for the activation function.

of the neuron. This process is illustrated in Figure D.1. The signal is propagated if $y(\mathbf{x})$ exceeds the 'threshold value' w_0. An easy example for the activation function is the Heaviside function $\Theta(x)$ which is 1 for $x \geq 0$ and -1 for $x < 0$. Obviously, Heaviside function is not differentiable in the classical sense in $x = 0$ and has value zero else. Therefore, the use of backpropagation algorithms is not possible (as we will see later). In general, a nonlinear function is used as activation function because it results in nonlinear capabilities of the neural network. One of the most common forms is the sigmoid which is a monitonic function that converges to finite values for $x \to \pm\infty$. Popular examples are the Fermi function (or logistic function)

$$F(y) = \frac{1}{1 + \exp^{-x}} \tag{D.2}$$

and the hyperbolic tangent. The Fermi function can be expanded introducing a temperature T which scales the x-axis:

$$F(y) = \frac{1}{1 + \exp^{-\frac{x}{T}}} \tag{D.3}$$

Thus, the shape of the Heaviside function (see Figure D.2) can be arbitrarily approximated (see Figure D.3).

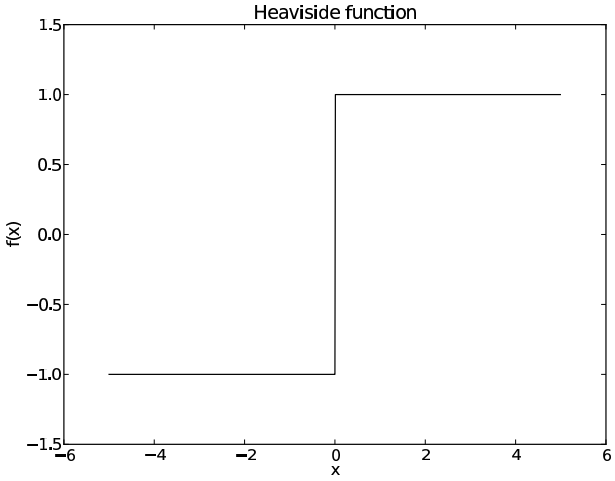

Fig. D.2: The Heaviside function.

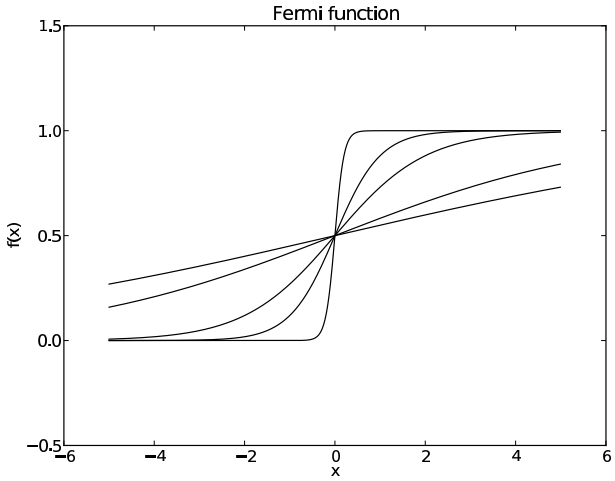

Fig. D.3: The Fermi function for which we vary the temperature.

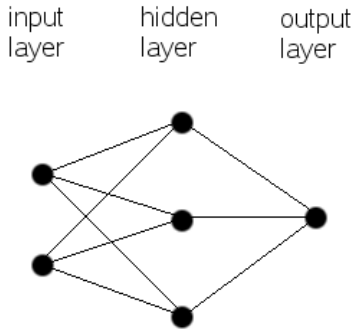

Fig. D.4: Example of a neural network.

D.2 The network topology

In the previous section we described the neuron which is the basic processing unit of a neural network. In this section we will focus on the topology of a neural network and how the signals are propagated. There are basically two possibilities for the topology of a neural network:

- *Feedforward networks* consist of an input layer, n hidden layers (that are invisible from outside) and one output layer. The connections are directed from a neuron to the next layer. Therefore, no feedback connections are present.
- *Recurrent networks* do contain feedback connections that are also directed towards any subsequent layer. Examples of recurrent networks can be found in Hopfield (1984).

The neural networks applied in this work belong to the class of feedforward networks. From now on, we will focus on these. A feedforward neural network generally consists of an arbitrary number of hidden nodes. However, it has been shown that one layer of hidden nodes is sufficient in order to approximate any nonlinear function with an arbitrary precision (Hartman *et al.* (1990), Funahashi (1989), Hornik (1989), Cybenko (1989)). The topology of such a feedforward neural network is shown in Figure D.4. A signal enters the input layer of the network. In the hidden layer, the incoming signals are weighted and propagated to the output layer (using the activation function). In the output layer, the same process is applied. The outgoing signals are then computed by

$$y = F\left(\sum_{i=1}^{m} w_i^{(2)} F\left(\sum_{j=1}^{n} w_{ij}^{(1)} + \theta_i^{(1)}\right) + \theta^{(2)}\right) \tag{D.4}$$

where n and m are the number of input and hidden nodes, respectively. The number of hidden nodes has to be adjusted to the specific problem.

D.3 How to train a neural network

So far we have discussed the processing units and the topology of a neural network. This chapter focuses on the algorithms which are needed in order to train the neural network. When we refer to a 'training', we mean the adjustment of the weights of the network. We use a so-called 'training sample', i.e., a set of events which we use to train the neural network. The objective of the training is the adjustment of the weights in such a way, that a set of inputs $\mathbf{x}$ produces the desired set of outputs $\mathbf{o}$. Different algorithms how to change the weights are well-known. One possibility is the use of a priori knowledge to set the weights explicitly. Another way is to feed the network in the training process with new patterns and letting it change the weights using a learning rule.

D.3.1 Error function

When adjusting the weights, we have to define an error measure which indicates the distance between current and correct output of the network. The training procedure tries to minimize this error. The optimal weights of the network are found when the error function is in its global minimum. The most commonly used error measure is the mean square error

$$E = \sum_i E^p = \frac{1}{2} \sum_p (d^p - y^p)^2 \tag{D.5}$$

where p sweeps over the set of input patterns, d^p is the actual output pattern and y^p is the output pattern of the network.

D.3.2 Gradient descent

The introduction of an error function translates the problem of finding optimal weights into a minimization problem. One of the most famous methods to find this minimum is the 'gradient descent' method. It may be applied if the gradient of the activation function can be easily defined, e.g. in the case of the logistic function (introduced in (D.2)). The idea is a change of the weights proportional to the negative gradient of the error function with respect to all weights

$$\Delta \mathbf{w} = -\eta \nabla E(\mathbf{w}) \tag{D.6}$$

Here η is the learning parameter and has to be fixed for the training. From (D.6) follows

$$dE = \nabla E \cdot d\mathbf{w} \approx \nabla E \cdot \Delta \mathbf{w} = -\eta |\nabla E(\mathbf{w})|^2 < 0 \tag{D.7}$$

Therefore, the error function indeed decreases if we change the weights as described by (D.6). We define the activation by

$$y_k^p = F(s_k^p), \tag{D.8}$$

in which

$$s_k^p = \sum_j w_{jk} x_j^p + \theta_k \tag{D.9}$$

For the error measure E^p we use the quadratic error function, defined in (D.5):

$$E^p = \frac{1}{2} \sum_k (d_k^p - y_k^p)^2 \tag{D.10}$$

where d_k^p is the desired output for unit k of the output layer when presenting pattern p to the network. The error of the whole training sample is defined by $E = \sum_p E^p$. In the next step we make use of the chain rule

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p} \frac{\partial s_k^p}{\partial w_{jk}} \quad . \tag{D.11}$$

and identify the second factor to be

$$\frac{\partial s_k^p}{\partial w_{jk}} = x_j^p \tag{D.12}$$

where we used D.9. The derivatives of the error function for pattern p become

$$\frac{\partial E^p}{\partial s_k^p} = -F'(s_k^p) \delta_k^p \tag{D.13}$$

with $\delta_k^p \equiv d_k^p - y_k^p$. Now we can combine (D.12), (D.13), (D.11) and (D.6)to the so-called *delta rule*:

$$\Delta^p w_{jk} = \eta F'(s_k^p) \delta_k^p x_j^p \tag{D.14}$$

D.3.3 Back-propagation

In case of multilayer neural networks, the gradient descent method cannot be applied directly and has to be generalized by the 'back-propagation' algorithm. If we applied the above mentioned delta rule directly to a multilayer neural network, we exclusively changed weights in the final layer. The weights of the previous layers would remain unchanged. The idea is to apply the chain rule again

$$\frac{\partial E^p}{\partial s_h^p} = \frac{\partial E^p}{\partial y_h^p}\frac{\partial y_h^p}{\partial s_h^p} = \mathcal{F}'(s_h^p)\frac{\partial E^p}{\partial y_h^p} \tag{D.15}$$

where h denotes the hidden node h. Furthermore, we get

$$\begin{aligned}\frac{\partial E^p}{\partial y_h^p} &= \sum_{o=1}^{N_o}\frac{\partial E^p}{\partial s_o^p}\frac{\partial s_o^p}{\partial y_h^p} \\ &= \sum_{o=1}^{N_o}\frac{\partial E^p}{\partial s_o^p}\frac{\partial}{\partial y_h^p}\sum_{j=1}^{N_h} w_{jo}y_j^p \\ &= \sum_{o=1}^{N_o}\frac{\partial E^p}{\partial s_o^p}w_{ho} \\ &= -\sum_{o=1}^{N_o} w_{ho}\mathcal{F}'(s_o^p)\delta_o^p\end{aligned} \tag{D.16}$$

If we substitute this into (D.11), we contain

$$\frac{\partial E^p}{\partial w_{jh}} = -\mathcal{F}'(s_h^p)\sum_{o=1}^{N_o} w_{ho}\mathcal{F}'(s_o^p)\delta_o^p\frac{\partial s_h^p}{\partial w_{jh}} \tag{D.17}$$

and

$$\Delta^p w_{jh} = \eta\mathcal{F}'(s_h^p)\sum_{o=1}^{N_o} w_{ho}\mathcal{F}'(s_o^p)\delta_o^p x_j^p \tag{D.18}$$

for the change in the weight between input layer and hidden layer. Using the simple delta rule for the weights between the hidden layer and the output layer leads to

$$\Delta^p w_{ho} = \eta\mathcal{F}'(s_o^p)\delta_o^p y_h^p \tag{D.19}$$

To derive now the total change in the weight, everything has to be summed up over the number of patterns p presented to the network.

D.4 Advanced learning techniques

In the previous sections, we discussed the basics of how to define and solve a nonlinear optimization problem in order to get the optimal weights of a neural network. There are still some fundamental problems which complicate the minimization process. The first problem is that nonlinear optimization problems in a high-dimensional space typically have a lot of local minima. Using gradient descent, it is possible to get trapped in a local minimum although there might exist a much deeper minimum. A second problem is the so-called 'network paralysis'. During the training of the network it might happen that one or more weights are adjusted to very large values. The derivative of the error function with respect to the large weight may be nearly zero. Thus, the large weight is not changed much, see (D.14), and the training process can come to a standstill. In this section we present some advanced techniques which can avoid such problems.

D.4.1 Learning per pattern

In each iteration of the back-tracking algorithm, one has to compute the gradient of the error function with respect to the weights. Therefore, one has to pass through the entire training sample. The weights are then adjusted using the *true* gradient. Alternatively, one can use single patterns to adjust the weights. This is called 'stochastic learning'.
There are two advantages using stochastic learning. First, the weights can be adjusted without looping through the entire data sample. Second, the weight vector is computed using just one pattern. Therefore, the weight vector changes much in each step and the probability of getting trapped in a local minimum is decreased.

D.4.2 Momentum

Training of a neural network, we have to choose an appropriate learning rate η. Using gradient decent, the change of the weights is proportional to η, see (D.14). Therefore, a large learning rate is prefered in order to have an algorithm which quickly finds a solution of the optimization problem. If the learning rate is too large, the stepsize could be too large and the weights might oscillate around a minimum or even diverge. The optimization

algorithm can be improved by adding a 'momentum' term which depends on the previous weight:

$$\Delta\mathbf{w}(t+1) = -\eta\nabla E(\mathbf{w}) + \alpha\Delta\mathbf{w}(t) \tag{D.20}$$

where α is a constant and t indexes the step in the training algorithm. In Figure D.5 we illustrate the descent in weight-space for different learning rates.

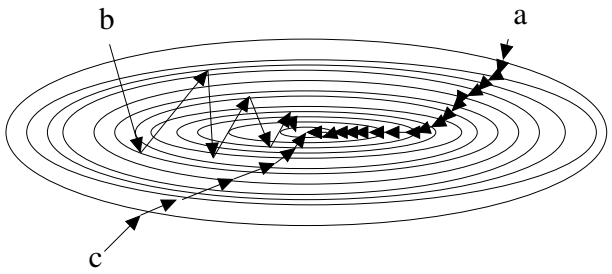

Fig. D.5: The descent of the weights in a learning algorithm. a) for a small learning rate; b) for a large learning rate; c) with a large learning rate and a momentum term added

D.4.3 Weight decay

Large weights can lead to outputs that are far beyond the largest outputs of the training sample. This can happen although the input vector is not significantly different from the input vectors of the training sample. Even discontinuities in the output are possible. This problem can be solved by penalizing large weights using a different error function

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{c}{2}\sum_i w_i^2 \tag{D.21}$$

where c is a constant. If the sum of the squared weights is dominating the error function, the update rule of the weights using gradient descent leads to

$$\Delta w_i = -\eta c w_i \tag{D.22}$$

or iterative

$$w_i(t+1) = w_i(t)(1-\eta c) \tag{D.23}$$

$$\Rightarrow \quad w_i(t) \quad = w_i(0)(1-\eta c)^t = w_i(0)\exp^{t\ln(1-\eta c)} \tag{D.24}$$

D.4.4 Pruning

Generally, a neural network should be as large as possible to be able to learn a broad class of problems. On the other hand, it should not overfit the data, i.e., learning noise or statistical unsignificant facts must be avoided. Therefore, if several nets fit the data equally well, the simplest one (i.e., the one having the fewest parameters) will on average have the best generalization capability. In order to penalise the complexity of a neural network, one can introduce the following term in the error function, see Weigend *et al.* (1990)

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \sum_i \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2} \tag{D.25}$$

where w_0 expresses a preference for fewer large (w_0 small) or many small weights (w_0 large). It can be used to delete connections having small weights. This process is called pruning.

D.5 Preprocessing of the inputs

The preprocessing of the input variables is one of the most important steps in the training of a neural network. If too many input variables are used, the neural network will have a lot of free parameters that have to be adjusted. Since an optimization problem in a multi-dimensional space has to be solved, the algorithm will converge slowly and probably end up in a local minimum although there might exist a much better minimum. Therefore, the significant input variables have to be carefully selected.
The second step is to transform the inputs to a set of variables with zero mean and variance one. This transformation leads to comparable inputs and a faster learning since the learning parameter η is balanced out globally for all variables. Furthermore, the activation function is a sigmoid and symmetric with respect to zero. Therefore it is better to have a working point near zero than a working point which is in the saturated region.
Another quite useful preprocessing step is the decorrelation of the input variables in order to make them linearly independent. The reason for the decorrelating is that high correlations in input variables lead to degenerecies, i.e., flat areas in the error surface. An adequate solution of such a degenerated optimization problem is much harder to find. Furthermore, the information provided in correlated variables is redundant. Thus, the variables do not improve the learning capabilities of the network.
The different steps are summarized in Figure D.6.

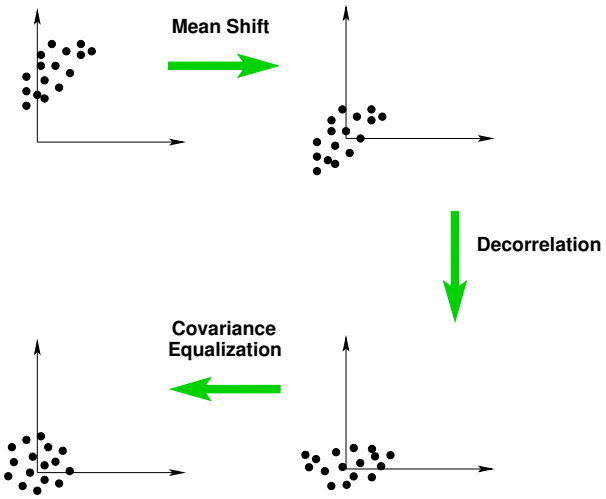

Fig. D.6: Illustration for the transformation of the inputs of a neural network.

D.6 Probability density reconstruction

Originally, neural networks were used for classification problems. In a classification problem, it is the objective of a neural network to derive the conditional probability $P(C_k|\mathbf{x}_i)$ of an input vector $\mathbf{x}_i$ belonging to class C_k. If the quadratic error function (defined in (D.5)) and the logistic function (defined in (D.2)) as activation function are used, the output can be interpreted as conditional probability, see Feindt (2004).

So far we have discussed how neural networks can deal with classification problems. In addition, we would like to use the neural network in order to reconstruct the continuous conditional probability density function of an event i. The idea is (see Feindt (2004)) to construct a network with **several** output nodes. If we use k output nodes, we have to do k binary classifications, namely *is the true value of the continuous output above the threshold value of node j or is it below the threshold value*. The *true value* is here transformed to its cumulative distribution function.

$$\tilde{t} = F(t) = \int_{min}^{t} f(t')dt' \qquad , \tag{D.26}$$

where t_{min} is the minimal value of the target in the training sample. Therefore we have $\tilde{t} \in [0, 1]$. From (D.26) follows that the propability density function is the derivative of the network outputs.

Let us give an example using 20 output nodes $\mathbf{o}$ with threshold values defined in table D.1. For an event with $\tilde{t} = 0.18$, the output nodes are

$$\mathbf{o} = (+1, +1, +1, +1, -1, -1, -1, -1, -1, -1, \\ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1) \tag{D.27}$$

where $+1$ and -1 represent signal and background repsectively. For an event with target $\tilde{t} = 0.19$ the output vector would be the same. Therefore, we have a discretization error which depends on the number of output nodes. This method can be improved if we allow continuous values for all output nodes. Thus, an event with $\tilde{t} = 0.18$ can be presented by

$$\mathbf{o} = (+1, +1, +1, +1, -0.8, -1, -1, -1, -1, -1, \\ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1) \tag{D.28}$$

and the event $\tilde{t} = 0.19$ would be represented by

$$\mathbf{o} = (+1, +1, +1, +1, -0.8, -1, -1, -1, -1, -1, \\ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1) \tag{D.29}$$

As we can see, continuous values for the output nodes lead to a smaller discretization error. In Figure D.7 we present an example of an output vector of a neural network with 20 output nodes. The outputs which represent the cumulative distribution function are smoothened by a cubic b-spline. The pdf is computed from the first derivative of the cdf, see Figure D.8.

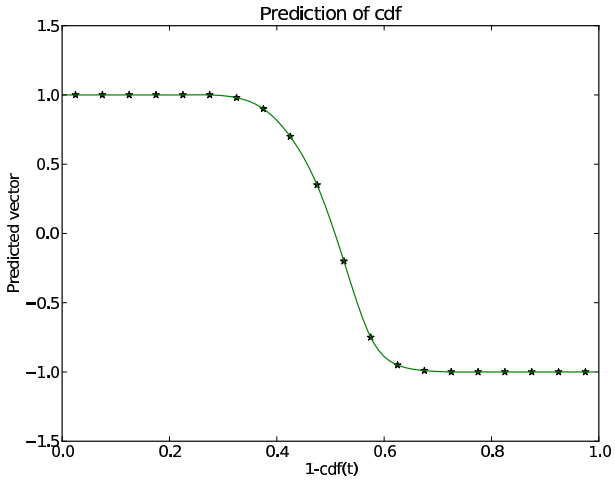

Fig. D.7: Example of a prediction of the cdf coming from a neural network with 20 output nodes.

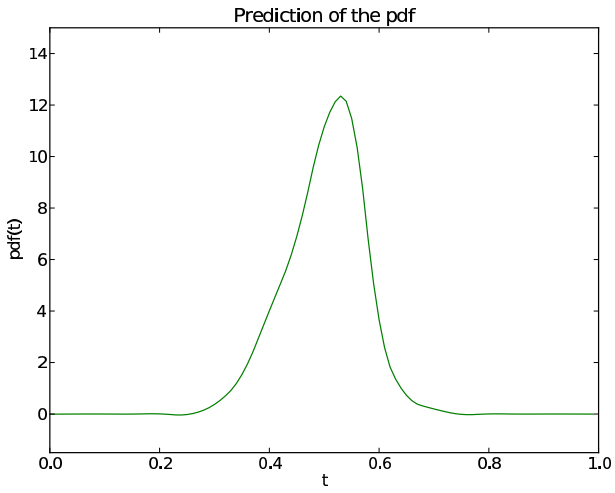

Fig. D.8: Example of a prediction of the pdf coming from a neural network with 20 output nodes.

number of node	threshold value
1	0.025
2	0.075
3	0.125
4	0.175
5	0.225
6	0.275
7	0.325
8	0.375
9	0.425
10	0.475
11	0.525
12	0.575
13	0.625
14	0.675
15	0.725
16	0.775
17	0.825
18	0.875
19	0.925
20	0.975

Table D.1: Example for the threshold values, represented by the output nodes.

E

Performance of the nn-models in the test sample

In this chapter we test the goodness-of-fit for the neural network models presented in part I. The Kolmogorov-Smirnov (KS) test can be used to test the null hypothesis of the empirical distribution to follow the suggested distribution. The test statistic is based on the maximal distance between the empirical and theoretical cdf:

$$KS = \sqrt{n} \sup |F_{emp.}(x) - F_{theo.}(x)| \tag{E.1}$$

where n is the number of events. The results of the KS-test for all neural network trainings are presented in table E.1.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
19861222	0.88	0.96
19870123	0.84	1.10
19870225	0.69	0.94
19870327	0.89	0.96
19870429	0.88	1.04
19870601	0.86	0.90
19870701	0.98	0.88
19870803	0.93	1.00
19870902	0.87	0.92
19871005	0.78	1.03
19871104	0.84	0.98
19871207	0.96	0.93
19880108	0.93	1.03
19880209	0.92	0.98
19880311	0.91	0.98
19880413	0.83	0.92
19880513	1.01	1.03
19880615	0.92	0.92
19880718	0.80	1.10
19880817	0.87	1.03
19880919	0.73	0.80
19881019	0.93	0.86
19881118	0.88	1.10
19881221	0.97	1.02
19890124	0.88	0.96
19890224	0.96	0.94
19890329	0.97	0.92
19890428	0.81	0.81
19890531	0.88	0.91
19890630	0.93	1.01
19890802	0.78	1.00
19890901	0.83	0.84
19891004	0.93	0.90
19891103	0.83	0.94
19891206	0.92	0.80
19900109	0.92	0.87
19900208	0.90	0.86
19900313	0.81	0.85
19900412	0.86	0.94

Table E.1: KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
19900515	1.01	0.79
19900615	0.98	0.83
19900718	0.90	0.82
19900817	0.91	0.99
19900919	0.91	0.74
19901019	0.92	0.87
19901120	0.92	0.80
19901221	0.88	0.82
19910124	0.91	0.84
19910226	0.94	0.94
19910328	0.97	0.84
19910430	0.78	0.98
19910531	0.92	0.91
19910702	0.90	0.86
19910802	0.92	0.96
19910904	0.88	1.00
19911004	0.85	0.93
19911105	0.84	0.80
19911206	0.77	0.92
19920109	0.98	1.05
19920210	0.85	0.94
19920312	0.89	0.87
19920413	0.81	0.91
19920514	0.91	0.90
19920616	0.86	0.98
19920717	0.67	0.85
19920818	0.81	0.77
19920918	0.71	1.07
19921020	0.96	0.99
19921119	0.94	0.80
19921222	0.93	0.89
19930125	0.88	1.02
19930225	0.78	1.05
19930329	0.88	0.81
19930429	0.75	0.78
19930601	0.80	0.83
19930701	0.87	0.88
19930803	0.80	0.91
19930902	0.93	0.90
19931005	0.84	0.93

KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
19931104	0.81	1.02
19931207	0.88	0.95
19940107	1.00	0.90
19940208	0.88	0.86
19940311	0.87	0.89
19940413	1.04	0.88
19940516	0.85	0.93
19940616	0.92	0.88
19940719	0.75	0.88
19940818	0.82	0.86
19940920	0.78	0.94
19941020	0.79	1.09
19941121	0.80	0.97
19941222	0.80	0.95
19950125	0.76	1.04
19950227	0.97	0.95
19950329	0.71	0.85
19950501	0.73	0.88
19950601	0.89	0.81
19950703	0.79	1.09
19950803	0.74	0.94
19950905	0.72	0.92
19951005	0.71	0.86
19951106	0.81	0.92
19951207	0.93	0.93
19960110	0.76	0.97
19960209	0.72	0.98
19960313	0.87	0.99
19960415	0.75	0.93
19960515	0.80	0.87
19960617	0.83	0.97
19960718	0.72	0.93
19960819	0.75	0.96
19960919	0.87	0.89
19961021	0.81	1.09
19961120	0.85	0.85
19961223	0.84	0.87
19970124	0.90	0.88
19970226	0.87	1.00
19970331	0.83	0.93

KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
19970430	0.73	0.97
19970602	0.84	1.06
19970702	0.61	0.95
19970804	0.94	0.93
19970904	0.91	0.98
19971006	0.92	0.83
19971105	0.78	0.80
19971208	0.84	1.02
19980109	0.87	0.90
19980211	0.83	0.97
19980316	0.79	0.93
19980416	0.84	0.82
19980518	0.87	0.94
19980618	0.82	0.97
19980721	0.77	0.88
19980820	0.87	1.02
19980922	0.81	0.78
19981022	0.82	0.79
19981123	0.84	0.85
19981224	0.95	0.82
19990128	0.78	0.92
19990302	0.83	0.89
19990401	0.92	0.88
19990504	0.73	0.94
19990604	0.78	0.83
19990707	0.76	0.93
19990806	0.73	0.85
19990908	0.74	0.86
19991008	0.86	0.83
19991109	0.86	0.80
19991210	0.80	0.95
20000112	0.83	0.92
20000214	0.76	0.83
20000316	0.88	0.87
20000417	0.98	0.88
20000518	0.88	0.83
20000620	0.71	0.90
20000721	0.79	0.80
20000822	0.78	0.72
20000922	0.90	0.76

KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
20001024	0.69	0.72
20001124	0.83	0.77
20001227	0.91	0.83
20010130	0.93	0.75
20010302	0.86	0.76
20010403	0.85	0.81
20010504	0.81	0.68
20010606	0.85	0.86
20010709	0.85	0.77
20010808	0.83	0.83
20010910	0.81	0.79
20011016	0.88	0.81
20011115	0.71	0.79
20011218	0.95	0.80
20020122	0.76	0.76
20020222	0.82	0.78
20020326	0.80	0.81
20020426	0.84	0.70
20020529	0.80	0.81
20020628	0.93	0.88
20020731	0.92	0.83
20020830	0.93	0.70
20021002	0.87	0.89
20021101	0.94	0.94
20021204	0.72	0.82
20030107	1.06	0.91
20030207	0.87	0.84
20030312	0.80	0.91
20030411	0.88	0.91
20030514	0.80	0.80
20030616	0.88	0.90
20030717	0.75	0.86
20030818	0.80	0.87
20030918	0.84	0.83
20031020	0.67	1.06
20031119	0.95	1.01
20031222	0.88	0.93
20040126	0.78	0.81
20040226	0.75	0.88
20040329	1.08	0.90

KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
20040429	0.82	0.83
20040601	0.82	0.83
20040702	0.75	0.85
20040804	0.80	0.79
20040903	0.88	0.91
20041006	0.85	0.72
20041105	0.82	0.91
20041208	0.90	0.95
20050110	0.82	0.83
20050210	0.85	0.94
20050315	0.98	0.82
20050415	0.88	0.96
20050517	0.78	0.89
20050617	0.86	0.91
20050720	0.93	0.89
20050819	0.83	0.83
20050921	0.85	0.78
20051021	0.86	0.88
20051122	0.93	0.92
20051223	0.97	0.95
20060127	0.82	0.80
20060301	0.95	0.93
20060331	0.87	0.81
20060503	0.87	0.85
20060605	0.79	0.80
20060706	0.92	0.87
20060807	0.77	0.82
20060907	0.90	1.09
20061009	0.87	0.79
20061108	0.88	0.80
20061211	0.89	0.77
20070116	0.99	1.03
20070215	0.90	1.04
20070320	0.81	0.87
20070420	0.82	0.71
20070522	0.84	0.76
20070622	0.93	0.75
20070725	0.74	0.99
20070824	0.77	0.99
20070926	0.74	0.90

KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

maximal date	KS EWMA-CTS-nn	KS GARCH-CTS-nn
20071026	0.76	0.78
20071128	0.79	0.85
20071231	0.88	0.74
20080201	0.91	0.79
20080305	0.88	0.75
20080407	0.99	0.77
20080507	0.91	0.98
20080609	1.06	0.83
20080710	0.87	0.74
20080811	0.83	0.73
20080911	1.01	0.85
20081013	0.89	0.93
20081112	0.77	0.77
20081215	0.72	0.73

KS statistic for the EWMA-CTS-nn and GARCH-CTS-nn model. The 95%-quantile is $q_{95} = 1.22$.

References

Achelis, S. B. (2000). *Technical Analysis from A to Z*. McGraw-Hill.

Anscombe, F. J. and Tukey, J. W. (1963). The examination and analysis of residuals. *Technometrics*, *5*(2), 141–160.

Bai, J. (1999). Likelihood ratio tests for multiple structural changes. *Journal of Econometrics*, *91*(2), 299–323.

Bai, J. and Perron, P. (1998). Estimating and testing linear models with multiple structural changes. *Econometrica*, *66*(1), 47–78.

Barndorff-Nielsen, O. E. and Shephard, N. (2001). Normal modified stable processes. *Economics Series Working Papers from University of Oxford, Department of Economics*, *72*.

Billingsley, P. (1968). *Convergence of Probability Measures*. John Wiley & Sons Inc.

Blobel, V. and Lohrmann, E. (1998). *Statistische und numerische Methoden der Datenanalyse*. Teubner Studienbuecher.

Bollerslev, T. (1986a). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*, 307–327.

Bollerslev, T. (1986b). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327.

Bollerslev, T. (1987). The review of economics and statistics. *The Review of Economics and Statistics*, *69*, 542–547.

Boyarchenko, S. I. and Levendorskiĭ, S. Z. (2002). *Non-Gaussian Merton-Black-Scholes Theory*. World Scientific.

Campbell, S. D. (2006). A review of backtesting and backtesting procedures. *Journal of Risk*.

Carr, P., Geman, H., Madan, D., and Yor, M. (2002). The fine structure of asset returns: An empirical investigation. *Journal of Business*, *75*(2), 305–332.

Chatfield, C. and Collins, A. J. (1980). *Introduction to multivariate analysis*. Chapman & Hall.

Colby, R. W. (2002). *The Encyclopedia Of Technical Market Indicators*. McGraw-Hill.

Cont, R. and Tankov, P. (2004). *Financial Modelling with Jump Processes*. Chapman & Hall / CRC.

Cowan, G. (1998). *Statistical Data Analysis (Oxford Science Publications)*. Oxford University Press, USA.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, *2*(4), 303–314–314.

Engle, R. F. (1982a). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, *50*(4), 987–1007.

Engle, R. F. (1982b). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, *50*(4), 987–1007.

Feindt, M. (2004). A neural bayesian estimator for conditional probability densities.

Fraenkle, J., Rachev, S. T., and Scherrer, C. (2010). Market impact measurement of a vwap trading algorithm. *submitted*.

Freund, R. J., Vail, R. W., and Clunies-Ross, C. W. (1961). Residual analysis. *Journal of the American Statistical Association*, *56*(293), 98–104.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Netw.*, *2*(3), 183–192.

Goldberger, A. S. (1961). Stepwise least squares: Residual analysis and specification error. *Journal of the American Statistical Association*, *56*(296), 998–1000.

Haag, C. (2003). *Praezissionsmessungen der Lebensdauer von identifizierten b-Hadronen mit dem DELPHI-Detektor*. Ph.D. thesis.

Hartley, H. O. (1961). The modified gauss-newton method for the fitting of non-linear regression functions by least squares. *Technometrics*, *3*(2), 269–280.

Hartman, E. J., Keeler, J. D., and Kowalski, J. M. (1990). Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, *2*(2), 210–215.

Hocking, R. R., Speed, F. M., and Lynn, M. J. (1976). A class of biased estimators in linear regression. *Technometrics*, *18*(4), 425–437.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*

of the United States of America, *81*(10), 3088–3092.

Hornik, K. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.

Huber, P. J. (1977). Robust methods of estimation of regression coefficients. *Series Statistics*, *8*(1), 41–53.

Irie, B. and Miyake, S. (1988). Untitled.

Johansen, T. A. and Foss, B. A. (1995). Identification of non-linear system structure and parameters using regime decomposition. *Automatica*, *31*(2), 321–326.

Kim, Y. S., Rachev, S. T., Bianchi, M. L., and Fabozzi, F. J. (2008a). Financial market models with Lévy processes and time-varying volatility. *Journal of Banking and Finance*, *32*(7), 1363–1378.

Kim, Y. S., Rachev, S. T., Bianchi, M. L., and Fabozzi, F. J. (2008b). A new tempered stable distribution and its application to finance. In G. Bol, S. T. Rachev, and R. Wuerth (Eds.), *Risk Assessment: Decisions in Banking and Finance*, Physika Verlag, Springer. 77–110.

Kim, Y. S., Rachev, S. T., Bianchi, M. L., and Fabozzi, F. J. (2009). Tempered stable and tempered infinitely divisible GARCH models.

Kim, Y. S., Rachev, S. T., Chung, D. M., and Bianchi, M. L. (2008c). The modified tempered stable distribution, GARCH-models and option pricing. *Forthcoming in Probability and Mathematical Statistics*.

Koponen, I. (1995). Analytic approach to the problem of convergence of truncated Lévy flights towards the gaussian stochastic process. *Physical Review E*, *52*, 1197–1199.

Kupiec, P. (1995). Techniques for verifying the accuracy of risk measurement models. *Journal of Derivatives*.

Mandelbrot, B. (1963). The variation of certain speculative prices. *The Journal of Business*, *36*(4), 394–419.

Mantel, N. (1970). Why stepdown procedures in variable selection. *Technometrics*, *12*(3), 621–625.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, *11*(2), 431–441.

Menn, C. and Rachev, S. T. (2005a). A garch option pricing model with [alpha]-stable innovations. *European Journal of Operational Research*, *163*(1), 201–209.

Menn, C. and Rachev, S. T. (2005b). Smoothly truncated stable distributions, GARCH-models, and option pricing.

Pope, P. T. and Webster, J. T. (1972). The use of an f-statistic in stepwise regression procedures. *Technometrics*, *14*(2), 327–340.

Protter, P. E. (2003). *Stochastic Integration and Differential Equations*. Springer, 2nd ed.

Quandt, R. E. (1958). The estimation of the parameters of a linear regression system obeying two separate regimes. *Journal of the American Statistical Association*, *53*.

Rachev (2003). *Handbook of Heavy Tailed Distributions in Finance (Handbooks in Finance)*. JAI Press.

Rachev, S. T., Fabozzi, F. J., and Menn, C. (2005). *Fat-Tailed and Skewed Asset Return Distributions : Implications for Risk Management, Portfolio Selection, and Option Pricing*. Wiley.

Rachev, S. T., Mittnik, S., Fabozzi, F. J., Focardi, M., and Jasic, T. (2007). *Financial Econometrics From Basics to Advanced Modeling Techniques*. The Frank J. Fabozzi Series. Wiley.

Rosinski, J. (2007a). Tempering stable processes. *Stochastic Processes and Their Applications*.

Rosinski, J. (2007b). Tempering stable processes. *Stochastic Processes and Their Applications*.

Scherer, M., Rachev, S. T., Kim, Y. S., and Fabozzi, F. J. (2010). A fft-based approximation of tempered stable and tempered infinitely divisible distributions.

Scherrer, C., Rachev, S. T., Feindt, M., and Fabozzi, F. (2010a). From the simple linear regression to the individualized linear regression. *in preparation*.

Scherrer, C., Rachev, S. T., Kim, Y. S., Feindt, M., and Fabozzi, F. (2010b). Using a neural network approach for backtesting methodologies for estimating and forecasting asset risk. *in preparation*.

Wahba, G. (1990). *Spline Models for Observational Data (C B M S - N S F Regional Conference Series in Applied Mathematics)*. Soc for Industrial & Applied Math.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1990). Generalization by weight-elimination with application to forecasting. In *NIPS-3: Proceedings of the 1990 conference on Advances in neural information processing systems 3*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 875–882.

Weigend, A. S. and Srivastava, A. N. (1995). Predicting conditional probability distributions: a connectionist approach. *International Journal of Neural Systems*, *6*(2), 109–118.

Zumbach, G. (2006). Backtesting risk methodologies from one day to one year, tech. report, risk metrics group.

Danksagung

Mein herzlicher Dank geht an

- meinen Betreuer Prof. Dr. Rachev, der es mir überhaupt erst ermöglichte in diesem Rahmen extern zu promovieren. Insbesondere die Freiheit in meinen Forschungsthemen, kombiniert mit ständiger Erreichbarkeit, führten zu einer guten und unkomplizierten Zusammenarbeit und interessanten Ergebnissen.
- an meinen externen Betreuer und Korreferenten Prof. Dr. Michael Feindt, der mich schnell davon überzeugen konnte, diese Arbeit extern in Zusammenarbeit mit der Firma Phi-T® anzufertigen. Die Höhepunkte einer hervorragenden Betreuung waren sicherlich die etlichen gemeinsamen Diskussionen, in denen ich unheimlich viel dazulernen durfte.
- Jochen Bossert, Geschäftsführer von Phi-T®, der mir die Finanzierung meiner Arbeit ermöglichte und mir in meinen Arbeitszeiten alle Freiheiten ließ.
- Dr. Christian Haag, der mich nicht nur von seiner Erfahrung als Fondsmanager profitieren ließ, sondern der sich vor allem auch in mathematischen Diskussionen und bei technischen Problemen stets hilfsbereit und aufgeschlossen zeigte.
- meinen Doktorandenkollegen Jan Fränkle für etliche gemeinsame Diskussionen innerhalb und außerhalb dieser Arbeit und die gute Zusammenarbeit bei gemeinsamen Projekten.
- Dr. Markus Kreer, der mir half, sowohl Veröffentlichungen als auch diese Arbeit mühsam sprachlich und strukturell zu verbessern und mich von seinen umfangreichen Literaturkenntnissen profitieren ließ.
- Dr. Bruno Daniel und Martin Hahn, die mir die Grundlagen der NeuroBayes Technologie näherbrachten, aber auch bei praktischen Problemen stets hilfsbereit waren.
- meine Doktorandenkollegen Michael Pieper und Alexander Beck, die stets aufgeschlossen für Diskussionen und das Probelesen meiner Dokumente waren.
- Dr. Martina Reber, insbesondere dafür, dass sie mich immer wieder motivierte zielgerichtet und schnell vorzugehen.
- Dr. Aaron Kim fuer die gute Zusammenarbeit und die pädagogisch hervorragende Einführung in für mich neue Themen.
- den Tischfußballkasten bei Phi-T®, der sich stets als aufgeschlossen gegenüber neuartigen Ideen präsentierte und mir immer motivierend zur Seite stand :-)
- alle anderen, mit denen ich während meiner Forschungszeit zusammenarbeiten durfte. Insbesondere geht mein Dank an alle meine Kollegen von Phi-T® für die familiäre Atmosphäre und die vielen interessanten und lustigen Gespräche innerhalb und außerhalb der Arbeit.
- meine Freundin Julia Bohnert für ihre mentale Unterstützung und ihr permanentes Interesse an meinen Forschungsthemen, und vor allem dafür, dass es sie gibt :-)
- meine Familie, die mir die Möglichkeit gab, mein Studium schnell und sorgenfrei zu absolvieren und stets sogar inhaltliches Interesse an meiner Arbeit zeigte, was mich immer wieder überraschte.