S Plus
For Financial Engineers [1]
-
PART II
-
The Dynamical Process Behind Financial Markets

Diethelm Würtz

Institut für Theoretische Physik
ETH Zürich

May 8, 2002

[1]This script collects material used in my lecture *Econophysics* held in SS 2002 at the "Insitute of Theoretical Physics" of ETH Zürich. These notes are thought for internal use only, please do not distribute! The software used in the script comes without any warranty!

.

Overview

Chapter 1 - Markets, Basic Statistics, Date and Time
1.1 Economic and Financial Markets
1.2 Brief Repetition from Probability Theory
1.3 Distribution Functions in Finance
1.4 Searching for Correlations and Dependencies
1.5 Hypothesis Testing
1.6 Calculating and Managing Calendar Dates

Chapter 2 - The Dynamical Process Behind Financial Markets
2.1 ARIMA Modelling: Basic Concepts of Linear Processes
2.2 GARCH Modelling: Mastering Heteroskedastic Processes
2.3 Regression Modelling from the Time Series Point of View
2.4 Neural Networks: Feedforward Connectionist Networks
2.5 Design and Implementation of an Intra-Daily Trading System

Chapter 3 - Beyond the Sample: Dealing With Extreme Values
3.1 Generalized Extreme Value Distribution
3.2 Point Processes - Generalized Pareto Distribution
3.3 Temporal Dependence: The Extremal Index

Chapter 4 - Pricing and Hedging of Options
4.1 Plain Vanilla Options
4.2 Heston Nandi Option Pricing
4.3 MC Simulation of Path Dependent Options

Appendix
An Introduction to R / S-plus

.

Contents

.

Chapter 2

The Dynamical Process Behind Financial Markets

If you can look into the seeds of time,
And say which grain will grow and which will not,
Speak.
Wiliam Shakespear, Macbeth.

Introduction

The second Chapter covers dynamical aspects of financial market data from the time series analysis point of view. The chapter is divided in five sections concerned with the basic concepts of univariate linear stochastic models, with the methodology of modelling heteroskedastic behavior, with the concepts of regressions analysis, with the approach of feedforward connectionist networks, and finally with the ideas behind technical trading models.

In Section 2.1 where we introduce the concept of autoregressive-moving average processes we deal with the general concepts of model selection, parameter estimation and diagnosis checking for ARIMA models. We do not go into details of this big machinery of linear time series modelling, we concentrate on the concepts in that sense, that we use this approach to remove the linear part from a financial time series. Having achieved this, we can concentrate on the investigation of the residuals of the linear model.

Volatility clustering is one of the central aspects which requires more sophisticated time series models. Section 2.2 is dedicated to these models. Modelling heteroskedastic effects with the GARCH family of models is a well accepted step in this direction.

Section 2.3 is devoted to regression analysis from the time series point of view. We present the concepts of linear and additive regression models and their generalizations. Further topics are the projection pursuit regression approach and modelling with multivariate adaptive regression splines.

In Section 2.4, a differently motivated point of view takes the time series modelling approach based on neural networks. A very prominent class of these models are the connectionist function approximators which we can use to model nonlinear time series analysis. We give an introduction to feedforward neural networks and show how we can use them for multivariate financial market time series analysis.

In Section 2.5 we introduce concepts of technical trading analysis and present trading models based on volatility adjusted time series.

2.1 ARMA Modelling: Basic Concepts of Linear Processes

Introduction

When we like to get insight into the dynamical behavior of a financial time series, it is useful to regard the observed series $(x_1, x_2, \ldots, x_T)$, as a particular realization of a stochastic process. The process will be the family of a random variable X_t defined on an appropriate probability space. The stochastic process can be described by a T-dimensional probability distribution. If we could assume joint normality of the distribution then the T means, $E[x_1], E[x_2], \ldots, E[x_T]$, the T variances $Var[x_1], Var[x_2], \ldots, Var[x_T]$ and the $T(T-1)/2$ covariances $Cov[x_i, x_j]$, $i < j$ would completely characterize the properties of the stochastic process. However, such an assumption is unlikely appropriate for most financial time series.

If normality cannot be assumed, but the process is taken to be *linear*, in the sense that the current value of the process is generated by a linear combination of previous values of the process itself and current and past values of any other related processes, then again the set of mentioned expectations would capture its major properties.

The procedure of using a single realization to infer the unknown parameters of a joint probability distribution is only valid if the process is *ergodic*, which roughly means that the sample moments for finite stretches of the realization approach their population counterparts as the length of the realization becomes infinite. We will assume from now on that the time series have this property.

Stationarity

One important assumption is that of stationarity, which requires the process to be in a particular state of "statistical equilibrium". A stochastic process is called *strictly stationary* if its properties are uneffected by a change of the time origin. In other words, the joint probability distribution at any set of times $t_1, t_2, \ldots, t_m$ must be the same as the joint probability distribution at times $t_1 + \tau, t_2 + \tau, \ldots, t_m + \tau$, where τ is an arbitrary shift in time. For $m = 1$ this implies that the marginal probability distributions do not depend on time, which in turn implies that, so long as $E[|x_t|^2] < \infty$, both the mean and variance of x_t, must be constant, i.e.

$$E[X_1] = E[X_2] = \ldots = E[X_T] = \mu \ , \tag{2.1}$$

and

$$Var[X_1] = Var[X_2] = \ldots = Var[X_T] = \sigma^2 \ . \tag{2.2}$$

Setting $m = 2$ *strict stationarity* implies that all bivariate distributions do not depend on time: Thus all covariances are functions only of the time shift (or lag) τ, i.e. for all τ

$$\begin{aligned} Cov[X_1, X_{1+\tau}] &= Cov[X_2, X_{2+\tau}] = \ldots = Cov[X_{T-\tau}, X_T] \\ &= Cov[X_t, X_{t-\tau}] = \gamma_\tau \ . \end{aligned} \tag{2.3}$$

In the following γ_τ denotes the autocovariances and $\rho_\tau = \gamma_\tau/\gamma_0$ the *autocorrelation function*, ACF, both of which only depend on the lag τ. A process is called *weakly stationary* if (2.1), (2.2), and (2.3) hold, i.e. first and second order moments of the process exist and are invariant through time. Note that strict stationarity (time invariance of distributions) always implies weak stationarity (time invariance of first and second order moments) whereas the converse does not hold. Assuming joint normality, however, both concepts coincide, since the normal distribution is completely determined by its first and second order moments.

Wold's Decomposition Theorem

A fundamental theorem in time series analysis is *Wold's decomposition theorem*. The theorem states that every weakly stationary, purely non-deterministic process $(x_t - \mu)$ can be written as a *linear filter*, i.e. a linear combination, of a sequence of uncorrelated random variables. By purely non-deterministic we mean that any linearly deterministic components have been subtracted from $(x_t - \mu)$.

The linear filter representation is given by

$$x_t - \mu = u_t + \psi_1 u_{t-1} + \psi_2 u_{t-2} + \ldots = \sum_{j=0}^{\infty} \psi_j u_{t-j} \ , \quad \psi_0 = 1 \ . \tag{2.4}$$

The u_t are a sequence of *iid* random variables drawn from a distribution with mean zero and variance $\sigma^2 < \infty$, and $Cov[u_i, u_{i-k}] = 0$ for all $k \neq 0$. The u_t are also often called *innovations* or a *white noise* process. Wold's decomposition underlies all the theoretical models of time series.

2.1.1 Stationary ARMA-Processes

The linear filter representation (2.4) is the origin of many realistic time series models resulting from particular choices of the ψ-weights.

First Order Autoregressive Processes: AR(1)

Taking $\mu = 0$ without loss of generality, choosing $\psi_j = \phi^j$, allows equation (2.4) to be written as

$$\begin{aligned} x_t &= u_t + \phi u_{t-1} + \phi^2 u_{t-2} + \ldots \\ &= u_t + \phi(u_{t-1} + \phi u_{t-2} + \ldots) \\ &= \phi x_{t-1} + u_t \ . \end{aligned} \tag{2.5}$$

This is known as a *first-order autoregressive process*, given the acronym AR(1). The backshift or lag-operator B can be introduced for notation convenience:

$$Bx_t = x_{t-1} \quad , \quad \ldots , \quad , \quad B^m x_t = x_{t-m} \ .$$

Rearranging equation (2.5) yields:

$$\begin{aligned} x_t &= (1-\phi B)^{-1} u_t \\ &= (1 + \phi B + \phi^2 B^2 + \phi^3 B^3 + ...) u_t \ . \end{aligned}$$

This linear filter representation will converge as long as $|\phi| < 1$, which is therefore the stationarity condition.

The moments and the ACF ρ_k of the AR(1)-process can be characterized as follows

$$\begin{aligned} E[x_t] &= 0 \ , \\ Var[x_t] &= \sigma^2(1 + \phi^2 + \phi^4 + ...) = \gamma_0 \ , \\ Cov[x_t, x_{t-k}] &= \gamma_k = \phi\gamma_{k-1} \ , \quad k > 0 \ , \\ \rho_k &= \frac{\gamma_k}{\gamma_0} = \phi^k \ . \end{aligned} \tag{2.6}$$

Since $|\phi| < 1$ the ACF ρ_k shows a pattern which is decreasing in absolute value, implying that the linear dependence of two observations x_t and x_s becomes weaker with increasing distance between t and s. Thus if $\phi > 0$, the ACF decays exponentially to zero, while if $\phi < 0$, the ACF decays in an oscillatory matter. Both decays are being slow if ϕ is close to the non-stationary boundaries of $+1$ or -1.

First Order Moving Average Processes: MA(1)

Alternatively assuming in equation (2.4) $\psi_j = -\theta$ and $\psi_j = 0, j > 1$ the so called *moving average* process of order 1, MA(1), is obtained:

$$\begin{aligned} x_t &= u_t - \theta u_{t-1} \\ &= (1 - \theta B) u_t \ . \end{aligned} \tag{2.7}$$

Low order moments of the process in (2.7) are easily seen to be:

$$\begin{aligned} E[x_t] &= 0 \ , \\ Var[x_t] &= (1 + \theta^2)\sigma^2 = \gamma_0 \ , \\ Cov[x_t, x_{t-k}] &= -\theta\sigma^2 \cdot \delta_{k0} = \gamma_k \ , \\ \rho_k &= -\theta\sigma^2 \cdot \delta_{k0} \ . \end{aligned}$$

Here, δ_{k0} is 1 for $k = 0$ and otherwise 0. Note, the two observations x_t and x_s generated by a MA(1) process are uncorrelated if t and s are more than one observation apart from each other.

Autoregressive Moving Average Processes: ARMA

Both processes considered so far, i.e. the AR(1) and MA(1) process, impose strong restrictions on the pattern of the corresponding autocovariance or autocorrelation function. More general patterns of linear dependencies are allowed by autoregressive or moving average models of higher order. The AR(p)- and MA(q)-model are defined as follows:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + u_t \quad (\text{AR(p)} - \text{processes}) \ , \tag{2.8}$$

and

$$x_t = u_t - \theta_1 u_{t-1} - \theta_2 u_{t-2} - \ldots - \theta_q u_{t-q} \quad (\text{MA(q)} - \text{processes}) \ . \tag{2.9}$$

A generalization of both time series models is obtained by combining the AR(p) and MA(q) processes defining an ARMA(p,q)-process:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + u_t - \theta_1 u_{t-1} - \theta_2 u_{t-2} - \ldots - \theta_q u_{t-q} \ . \tag{2.10}$$

Using the lag operator B equation (2.10) may be written as follows

$$\begin{aligned} (1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p) x_t &= (1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_q B^q) u_t \ , \\ \phi(B) x_t &= \theta(B) u_t \ . \end{aligned} \tag{2.11}$$

There is no simple general expression for the variance, covariance and autocorrelation function of a stationary ARMA(p,q) model. These expressions are solutions to difference equations that cannot easily be solved by inspection. However, there are simple formulas for the variance of he AR(p) model and the MA(p) model. For the AR(p) model we have

$$Var[x_t] = \frac{\sigma_u^2}{\phi_1 \rho_1 - \cdots - \phi_p \rho_p} \ .$$

and for the MA(q) we have

$$Var[x_t] = (1 + \theta_1^2 + \cdots + \theta_q^2) \sigma_u^2 \ .$$

Concerning the ACF, in general an AR(p) process is described by a correlation function that has infinite extent and is represented by oscillating damped exponentials, and in general a MA(q) process cuts after leg q. The ACF of the resultant ARMA(p,q) will eventually follow the same pattern as that of an AR(p) process after $q - p$ initial values.

A process of low order which is already characterized by a flexible pattern of its autocorrelation function, e.g. in comparison to the AR(1) or to the MA(1) process, is the ARMA(1,1)-process

$$x_t - \phi x_{t-1} = u_t - \theta u_{t-1},\ |\phi| < 1,\ |\theta| < 1\ . \tag{2.12}$$

Autocovariances of order k are easily determined for the ARMA(1,1) by multiplying equation (2.12) with x_{t-k} and taking expectations:

$$E[x_t x_{t-k} - \phi x_{t-1} x_{t-k})] = E[u_t x_{t-k} - \theta u_{t-1} x_{t-k})]\ .$$

Furthermore, the following results can be derived

$$\begin{aligned} \gamma_k &= \phi\theta_{k-1}\ ,\quad k > 1\ \ , \\ \gamma_0 - \phi\gamma_1 &= \sigma^2(1 - \theta(\phi - \theta))\ \ ,\quad \text{from } k = 0\ , \\ \gamma_1 - \phi\gamma_0 &= -\theta\sigma^2\ \ ,\quad \text{from } k = 1\ . \end{aligned}$$

Turning to the autocorrelation function one obtains:

$$\rho_1 = \frac{(1 - \phi\theta)(\phi - \theta)}{1 + \theta^2 - 2\phi\theta}\ , \qquad \text{and} \qquad \rho_k = \phi\rho_{k-1}. \tag{2.13}$$

This equation shows that the autoregressive coefficient governs how the ACF dies out whereas the moving average parameter is important for the initialization of the ACF ρ_1.

Partial Autocorrelation Function: PACF

Since all AR processes have ACFs that damp out it can be difficult to distinguish between processes of different orders. To aid with such discrimination we may use the partial autocorrelation function, named briefly PACF. Whereas the sample ACF $\widehat{\rho}_k$ is computed from a time series realization $x_1, x_2, \ldots, x_n$ and gives the observed correlation between pairs of observations (x_t, x_{t+k}), having mean $\overline{x}$ and separated by time spans k

$$\widehat{\rho}_k = \frac{\sum_{t=1}^{n-k}(x_t - \overline{x})(x_{t-k} - \overline{x})}{\sum_{t=1}^{n-k}(x_t - \overline{x})^2},\quad k = 1, 2, \ldots\ , \tag{2.14}$$

the sample PACF $\widehat{\phi}_{kk}$ gives the correlations between k-separated ordered pairs (x_t, x_{t+k}) with the effect of intervening observations $x_{t+1}, x_{t+2}, \ldots, x_{t+k-1}$ removed. The sample PACF can be computed as a function of the sample ACF

$$\widehat{\phi}_{11} = \widehat{\rho}_1 \tag{2.15}$$

$$\widehat{\phi}_{11} = \frac{\widehat{\rho}_k - \sum_{j=1}^{k-1}\widehat{\phi}_{k-1j}\widehat{\rho}_{k-j}}{1 - \sum_{t=1}^{k-1}\widehat{\phi}_{k-1j}\widehat{\rho}_j}\quad k = 2, 3, \ldots \tag{2.16}$$

where

$$\widehat{\phi}_{kj} = \widehat{\phi}_{k-1j} - \widehat{\phi}_{kk}\widehat{\phi}_{k-1k-j} \quad k = 3, 4, \ldots \quad j = 1, 2, \ldots, k-1 \ .$$

This method of computing the sample PACF is based of a set of equations known as the Yule-Walker equations that give $\rho_1, \rho_2, \ldots, \rho_k$ as a function of $\phi_1, \phi_2, \ldots, \phi_k$. Each estimated PACF coefficient $\widehat{\phi}_{kk}$ is an estimate of the corresponding model-based true PACF ϕ_{kk}.

Splus - ARMA Simulating Processes

Splus offers a whole suite of functions for modelling ARIMA processes. The first we like to use is `arima.sim()` for the simulation of ARMA processes.

The *required argument* `model` for the Splus function `arima.sim()` is a list of parameters specifying an ARIMA model. The components of the model argument should be some or all of: `order`, `period`, `ar`, `ma`, `ndiff`, `ar.opt`, `ma.trans`, and `ma.opt`. `order` is a vector of length 3 specifying the order of an ARIMA(p,d,q); the elements of the vector are (p, d, q), where `p` and `q` are the order of the autoregressive and moving average operators and `d` is the number of differences. `period` specifies the period of the ARIMA operators; e.g., a seasonal operator for monthly data would have a period of 12 (the default is 1). `ar` and `ma` are vectors of initial values for autoregressive and moving average coefficients respectively (if not provided, the initial values to the optimizer are set to 0). `ndiff` is an alternative way to specify the number of differences. A model can be specified using either `order` or `ar`, `ma`, and `ndiff`, or both (but they must agree if both are specified). The model doesn't explicitly allow for a non-zero mean of the series, though the estimation is invariant to the mean of the data for many of the possible models; if you estimate an AR model, you must subtract the mean yourself or use the `xreg` argument in order to get sensible estimates. If `ma.trans` is TRUE (the default), the moving average coefficients will be transformed before passing them to the optimizer, ensuring invertibility of the model. `ar.opt` and `ma.opt` are logical vectors of length `p` and `q` respectively. If the ith element is TRUE, then the optimizer will optimize over the ith autoregressive or moving average coefficient. By default, the vectors are TRUE. This option is useful to fit models in which some low order coefficients are set to 0. The likelihood is conditioned on `n.cond` observations.

Several *optional arguments* can be added: `n`, the length of the series to be simulated (optional if `innov` is provided). `innov`, a univariate time series or vector of innovations to produce the series. If not provided, `innov` will be generated using rand.gen. Missing values are not allowed. `n.start`, the number of start-up values discarded when simulating non-stationary models. The start-up innovations will be generated by `rand.gen` if `start.innov` is not provided. `start.innov`, a univariate time series or vector of innovations to be used as start up values. Missing values are not allowed. `rand.gen`, a function which is called to generate the innovations. Usually, `rand.gen` will be a random number generator. `xreg`,a univariate or multivariate time series, or a vector, or a matrix with univariate time series per column. These will be used as additive regression variables. `reg.coef`, a vector of regression coefficients corresponding to `xreg`. `...` additional arguments may be passed to `rand.gen`.

The *returned value* is an univariate time series, the simulated series.

Example: ARMA Modelling, Simulating Processes - `xmpArimaSimulation`:

Now use this Splus function and simulate the following AR, MA and ARMA processes with a length of 1000 data points: AR(1) processes with $\phi = \pm 0.5$, AR(2) processes with $\phi_1 = \pm 0.5$ and $\phi_2 = 0.3$, MA(1) processes with $\theta = \pm 0.8$, and ARMA(1,1)processes with $\phi = \pm 0.5$, and $\theta = 0.8$.

```
x <- arima.sim(n=1000, model=list(order=c(1, 0, 1), ar=0.5, ma=0.8))
plot(1:length(x), x, type="l", main="ARMA(1,1): +0.5, +0.8")
```

Splus - ACF

The standard Splus function `acf()` estimates and displays autocovariance, autocorrelation or partial autocorrelation functions.

The *required argument* is `x`, a time series vector (or a matrix or a regular or a calendar time series). Missing values are allowed only at the beginning or end of series. If x is a matrix, rows are treated as time points and columns as univariate series.
The *optional arguments* include `lag.max`, the maximum number of lags at which to estimate the autocovariance. If this is not supplied, it is a number proportional to the logarithm of the length of the series. `type`, a character string: "covariance" to estimate the autocovariance function, "correlation" for the autocorrelation function, or "partial", if the partial autocorrelation function is desired. The start of one of the strings will suffice. `plot`, is a logical flag, if TRUE, the autocovariance or autocorrelation function between pairs of univariate series will be plotted in an array of at most 5 by 5 plots per page. If type is "correlation" or "partial" approximate 95% confidence limits are drawn on the plots.
The *returned value* is a list with the following components: `acf`, a three-dimensional array containing the autocovariance or autocorrelation function estimates. acf[i,j,k] is the covariance (or correlation) between the j-th series at time t and the k-th series at time t+1-i. `lag`, an array the same shape as acf containing the lags (as fractions of the sampling period) at which acf is calculated. If $j > k$ and $i > 1$, then lag[i,j,k] is negative. `n.used`, the number of observations in which no missing values occur. `type`, a character string indicating the type of function, "covariance", "correlation" or "partial". `series`, the name of x, including transformations.

Example: ARMA Modelling, Simulating Processes, cont. - `xmpArimaSimulation`:

Now, plot with the help of the Splus function `acf()` the sample autocorrelation function and the sample partial autocorrelation function. The results are shown in figure 2.1.1.

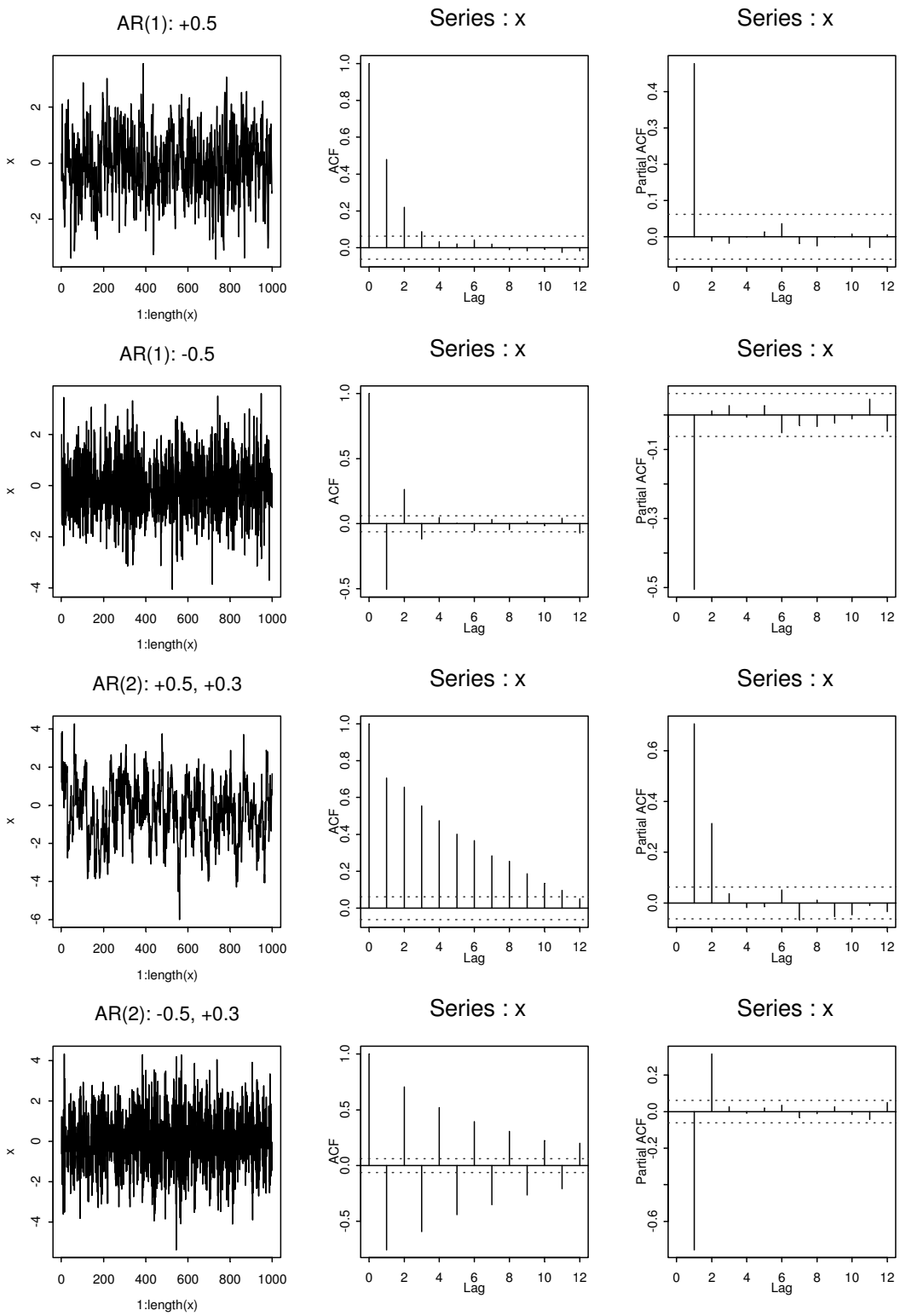
```
acf(x, lag.max=12)
acf(x, lag.max=12, type="partial")
```

Example: ARMA Modelling, Investigating the True ACF/PACF - `xmpArimaTrueAcf`:

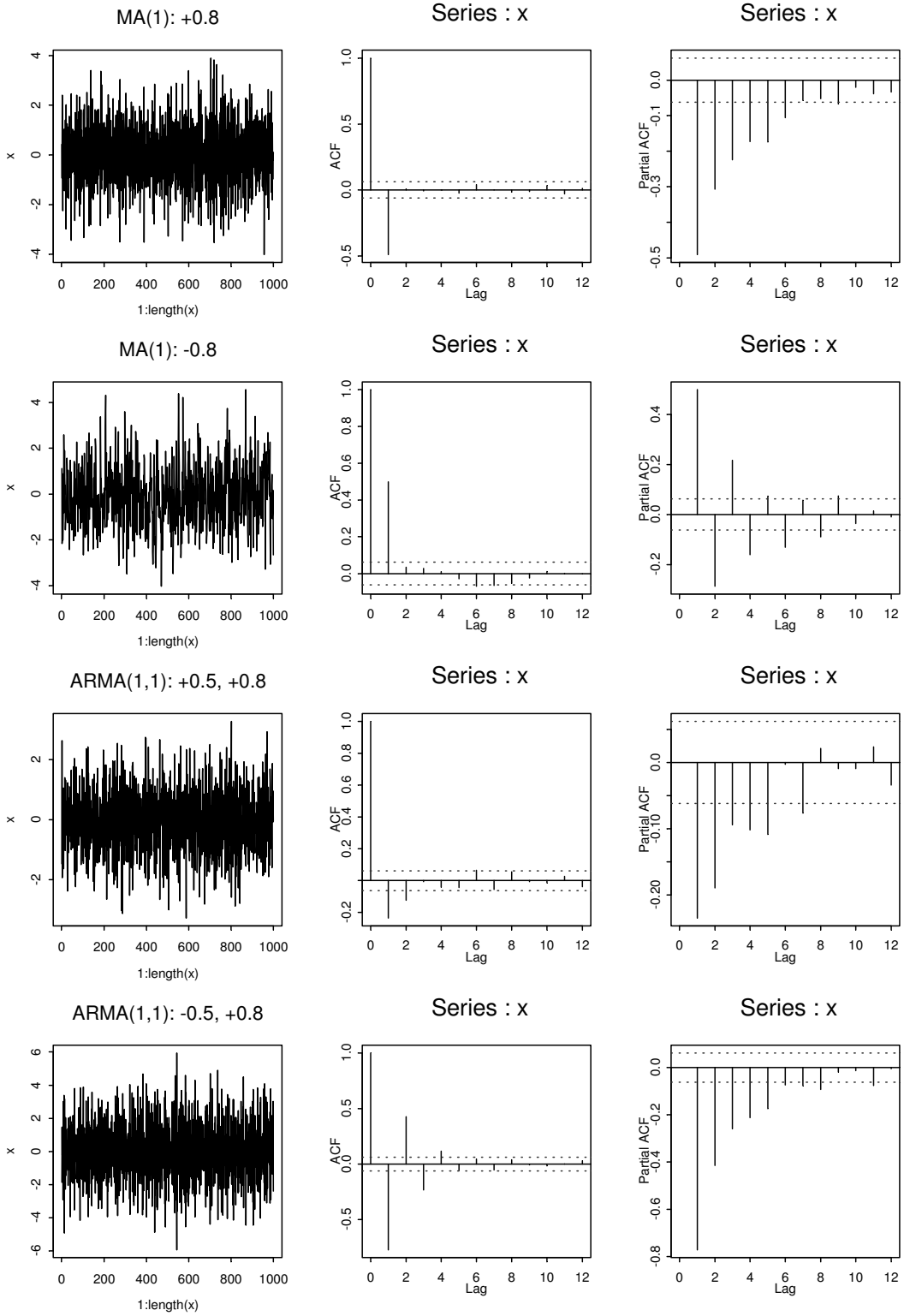
Compare the sample ACF / PCAF with the true ACF / PACF for two simulated AR(2) processes of length 1000 with $\pi_1 = \pm 0.5$ and $\phi_2 = 0.3$, respectively. Use the Splus function `trueacf()` from the `fSeries` library. The results are shown in figure 2.1.2. The graphs for the AR(2) models were produced by the following Splus commands:

```
# First Model:
  model <- list(ar=c(+0.5, 0.3))
  x <- arima.sim(n=1000, model)
  plot(1:length(x), x, type="l", main="AR(2): +0.5, 0.3")
  acf(x, lag.max=12)
  trueacf(model, lag.max=12)
  acf(x, lag.max=12, type="partial")
  trueacf(model, lag.max=12, type="partial")
# Second Model:
  model <- list(ar=c(-0.5, 0.3))
  x <- arima.sim(n=1000, model)
  plot(1:length(x), x, type="l", main="AR(2): -0.5, 0.3")
  acf(x, lag.max=12)
  trueacf(model, lag.max=12)
  acf(x, lag.max=12, type="partial")
  trueacf(model, lag.max=12, type="partial")
```

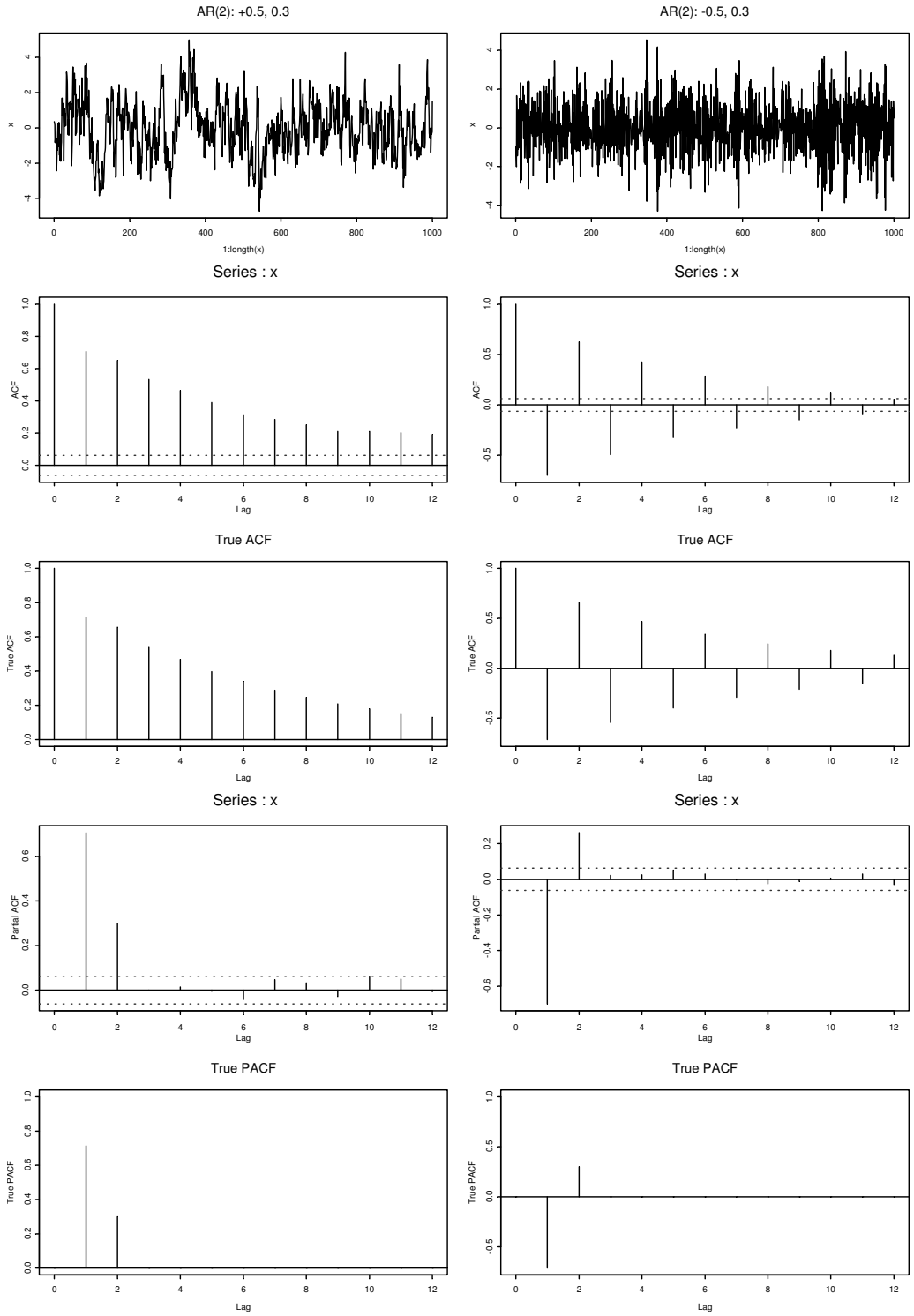
The models show an exponentially decaying ACF, whereas the second model's ACF is oscillating. As expected, the PACF shows the first two values to be significant,



■ Figure 2.1.1 shows for various AR, MA and ARMA time series models (left column) together with the ACF (middle) and PACF (right) From top to bottom we have two AR(1) processes with $\phi = \pm 0.5$ followed by two AR(2) processes with $\phi_1 = \pm 0.5$ and $\phi_2 = 0.3$.



■ Figure 2.1.1 cont. From top to bottom we have two MA(1) processes with $\theta = \pm 0.8$, and two ARMA(1,1) processes with $\phi = \pm 0.5$, and $\theta = 0.8$.



■ Figure 2.1.2 compares for two AR(2) time series models with $\phi_1 = \pm 0.5$ and $\phi_2 = 0.3$ the sample and the true ACFs and PACFs.

Stationarity and Invertibility Conditions

x_t is *stationary* if the roots of the characteristic polynomial $\phi(z)$ have moduli which are larger than one:

$$\phi(z) = (1 - \phi_1 z - \phi_2 z^2 - \ldots - \phi_p z^p) \neq 0 \quad \text{for} \quad |z| < 1 \ . \tag{2.17}$$

For a stationary ARMA(p,q) process the polynomial $\phi(B)$ can be inverted such that x_t has a moving average representation of infinite order: $x_t = \phi^{-1}(B)\theta(B)u_t$.

An ARMA(p,q) process is called *invertible* if the roots of $\theta(z)$ have moduli which are larger than one:

$$\theta(z) = (1 - \theta_1 z - \ldots - \theta_q z^q) \neq 0 \quad \text{for} \quad |z| < 1 \ . \tag{2.18}$$

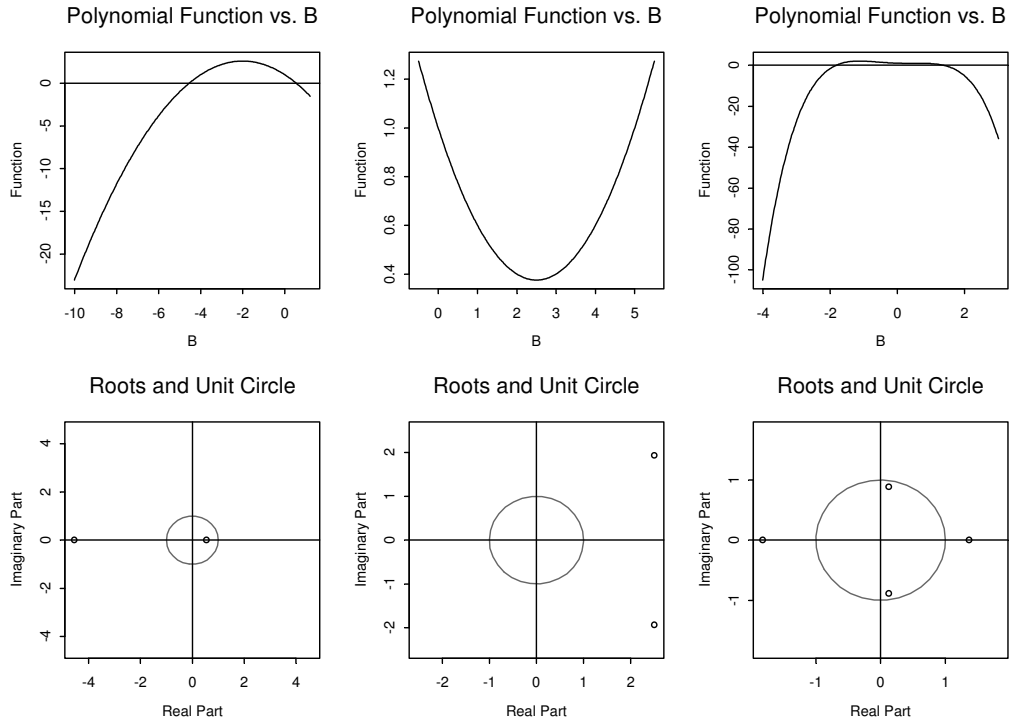
In this case x_t has an autoregressive representation of infinite order: $\theta^{-1}(B)\phi(B)x_t = u_t$.

Example: ARMA Modelling, Stationarity and Invertibility - `xmpArimaRoots`:

Let us inspect the Splus function `arma.roots()` from the `fSeries` library which can be used to compute and to display the roots of an AR or an MA polynomial.

```
"arma.roots" <-
function(coefficients, nplot=400, digits=4) {
    # Finds roots of the polynomial:
      root <- polyroot(c(1, -coefficients))
      real.root <- Re(root)
      im.root <- Im(root)
    # Plot polynomial function:
      xrange <- range(real.root)
      xrange <- c(xrange[1]-1.2*abs(xrange[1]), xrange[2]+1.2 * abs(xrange[2]))
      xplot <- seq(xrange[1], xrange[2], length = nplot)
      fpoly <- 1
      for (i in 1:length(coefficients)) {
        fpoly <- fpoly - xplot^i * coefficients[i] }
      plot(xplot, fpoly, type = "l", xlab = "B", ylab = "Function")
      title(main = "Polynomial Function vs. B")
      abline(h = 0)
    # Plot roots and unit circle:
      distance <- sqrt(real.root^2 + im.root^2)
      root.mat <- cbind(round(real.root, digits = digits),
        round(im.root, digits = digits), round(distance, digits = digits))
      dimnames(root.mat) <- list(1:nrow(root.mat), c("re", "im", "dist"))
      size.limit <- max(abs(real.root), 1.5, abs(im.root))
      plot(root, xlim = c( - size.limit, size.limit),
        ylim = c( - size.limit, size.limit), xlab = "", ylab = "")
      symbols(0, 0, circles = 1, add = T, inches = F, col = 6)
      abline(h = 0)
      abline(v = 0)
      title("Roots and Unit Circle", xlab = "Real Part", ylab = "Imaginary Part")
    # Return result:
      root.mat}
```

Now use this function to find the roots of the polynomials $(1 - 1.6B - 0.4B^2)$, $(1 - 0.5B - 0.1B^2)$, and $(1 - 0.5B + 0.9B^2 - 0.1B^3 - 0.5B^4)$.



■ Figure 2.1.3 displays for three polynomials the polynomial functions versus B (upper row) and the unit roots together with the unit circle (lower row). The polynomials from left to the right are: $(1 - 1.6B - 0.4B^2)$, $(1 - 0.5B - 0.1B^2)$, and $(1 - 0.5B + 0.9B^2 - 0.1B^3 - 0.5B^4)$.

```
arma.roots(c(1.6,0.4))
        re  im    dist
1   0.5495   0  0.5495
2  -4.5495   0  4.5495

arma.roots(c(0.5, -0.1))
    re      im     dist
1  2.5   1.9365  3.1623
2  2.5  -1.9365  3.1623

arma.roots(c(0.5, -0.9, 0.1, 0.5))
        re       im    dist
1   0.1275   0.8875  0.8966
2   0.1275  -0.8875  0.8966
3  -1.8211   0.0000  1.8211
4   1.3662   0.0000  1.3662
```

The graphs are displayed in figure 2.1.3. The polynomial in the first example has two real roots, one inside and the other outside of the unit circle. Thus if this were an MA (AR) polynomial the model would be noninvertible (nonstationary). The polynomial in second example has two imaginary roots, both of which are outside of the unit circle. Thus if this were an MA (AR) polynomial the model would be invertible (stationary). In the third example the roots are inside the unit circle implying noninvertibilty in the case of a MA model and nonstationarity in the case of an AR model.

Mean Stationarity and Differencing: ARIMA

ARMA models are most useful for predicting stationary time series. These models can be generalized to ARIMA models that have integrated random walk type behavior. These models are useful for describing certain kinds of nonstationary behavior.In particular, an integrated ARIMA model can be changed to a stationary ARMA model using a differencing operation.Then we fir an ARMA model to the differenced data. Differencing involves calculating successive changes in he values of a data series.

To difference a time series,we define a new variable Y_t which is he change in X_t from one time period to the next; i.e.

$$Y_t = (1 - B)X_t = X_t - X_{t-1} \ , \quad t = 2, 3, \ldots, n \ . \tag{2.19}$$

Y_t is called he first difference of X_t. We can also look at differencing from the other side. In particular, rewriting the above equation and using successive resubstitution, we end up after n steps with

$$X_t = X_{t-n} + Y_{t-n-1} + \cdots + Y_t \ . \tag{2.20}$$

This shows why we would say that a model with a $(1 - B)$-differencing term is *integrated*. Then in general

$$Y_t = (1 - B)^d X_t \tag{2.21}$$

is a d-th order regular difference. For financial time series (usually the log prices) that have integrated or random-walk-type behavior, first differencing leading to log returns, is usually sufficient to produce a time series with a stationary mean.

Including Deterministic Components

In the presentation of theoretical models given above a deterministic component was always excluded. The generalization of stationary ARMA(p,q)-processes allowing for a non-zero mean, however, is straightforward. Augmenting the stationary process (2.10) with $\nu \neq 0$ one obtains

$$\phi(B)x_t = \nu + \theta(B)u_t \ .$$

Inversion of $\phi(B)$ immediately yields the expectation of x_t,

$$\mu = E[x_t] = \phi(B)^{-1}\nu \ .$$

Note that if $\phi(B) = 1$ which is the pure MA(q) model, one has $\mu = \nu$.

2.1.2 Parameter Estimation, Model Selection and Diagnosis Checking

Performing time series analysis one usually tries to remove the linear dependencies first. This is achieved by fitting an AR, MA or ARMA process to the observed data. So there is first the problem of finding the correct order of the model - *model identification* - second, of fitting the parameters ϕ and θ - *parameter estimation* - third, of selecting the most likely model - *model selection* - and fourth, of checking the diagnostic properties of the selected model - *diagnosis checking*.

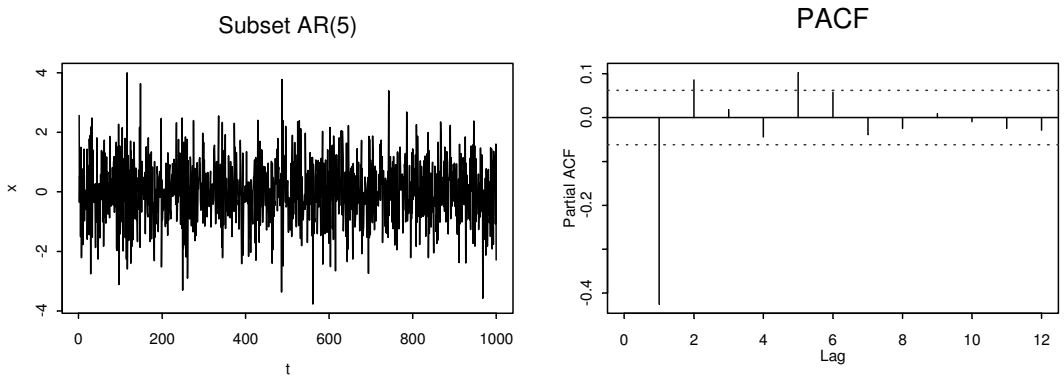
Model Identification

One of the basic tools of the time series analyst is the plot of the (estimated) autocorrelations ρ_k and partial autocorrelations ϕ_{kk} against the lag k. In theory, the shape of this plot can help to discriminate between competing linear models. It is an usual practice in time series analysis to try first to identify from summaries of the data one or just a few models that might have generated the data. As an example, if a process is ARMA(p, q), then $\rho_k = \theta^k$ for k large, with $|\theta| < 1$, but if $p = 0$, $\rho_k = 0$ for $k \geq q + 1$ so that the autocorrelations can, theoretically, help on decide if $p > 0$ and, if not, to choose the value of q.

Example: ARMA Modelling, Model Identification - `xmpArimaIdentification`:

First use the standard Splus function `arima.sim(model,...)` to simulate an AR(5) subset model with parameters $\phi_1 = -0.4$, $\phi_2 = 0.1$, and $\phi_5 = 0.1$. Identify the model by investigating the PACF with the help of the `acf()` function.

```
# Simulate the time series:
    x <- arima.sim(n=1000, n.start=100, rand.gen=rnorm,
        model=list(order=c(5, 0, 0),
        ar.opt=c(T, T, F, F, T), ar=c(-0.40, 0.1, 0, 0, 0.1)) )
# Plot the time series:
    plot(1:length(x), x, type="l", main="Subset AR(5)")
# Identify the process through the PACF:
    pacf <- acf(x, lag.max=12, type="partial")
```



■ Figure 2.1.4 displays to the left the subset AR(5) time series model with parameters $\phi_1 = -0.4$, $\phi_2 = \phi_5 = 0.1$ and to the right the associated PACF. Note, that the PACF is consistent with a subset AR(5) model with nonzero parameters ϕ at positions 1, 2 and 5.

Parameter Estimation

All the observations that can be made in a time series analysis are samples $(X_t)_{t=1,\ldots,N}$ of a certain length N. An ARMA model is described by the parameter vector

$$\Theta = \left(\sigma_u^2, \mu, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q\right) \ , \tag{2.22}$$

where σ_u^2 denotes the variance of the residuals and μ the mean of the process. A stochastic model with a parameter vector Θ determines a probability density $p\left(X \mid \Theta\right)$ on a set of time series. We want to estimate the value of Θ for a given fixed time series X_t. The likelihood function is defined as follows

$$L\left(\Theta \mid X\right) = p\left(X \mid \Theta\right) \ . \tag{2.23}$$

We think that the value of Θ with the largest likelihood function is the most reasonable, because in this case the probability to produce the observed data becomes maximal. If we assumed the u's are normally distributed with mean-zero and variance (conditional on past data) σ_u^2, the likelihood function is proportional to

$$\left(\sigma_u^2\right)^{-T/2} f(\chi) \exp\left[-S(\chi, X_1, X_2, ..., X_t)/ - 2\sigma_u^2\right] \ , \tag{2.24}$$

where χ contains the parameters in $\phi\left(B\right)$ and $\theta\left(B\right)$.

In more detail, the essential idea is the decomposition of prediction errors. We can factorize the joint density of $(X_1, \ldots, X_T)$ as

$$f(X_1, \ldots, X_T) = f(X_1) \prod_{t=2}^{T} f(X_t | X_1, \ldots, X_{t-1}) \ .$$

Suppose the conditional distribution of X_t given $(X_1, \ldots X_{t-1})$ is normal with mean $\widehat{X}_t$ and variance σ_{t-1}^2, and suppose also that X_1 is normal $N(\widehat{X}_1, \sigma_0)$. Here $\widehat{X}_t$ and σ_{t-1} are functions of the unknown parameters $\phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$ and the data. The log likelihood becomes

$$-2logL = -2logf = \sum_{t=1}^{T} \left[log(2\pi) + log(\sigma_{t-1}^2) + \frac{x_t - \widehat{x}_t}{\sigma_{t-1}^2} \right] \ . \tag{2.25}$$

We can minimize this with respect to $\phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$ to fit the ARMA(p,q) process. In practice the log likelihood $-2logL$ is modified to sum only over the range $\{fm+1, \ldots, T\}$, where m is small. For example for an AR(p) model choose $m = p$, so $\widehat{x}_t = \Sigma_{r=1}^{p} \phi_r x_{t-r}$, $t > m+1$, and $\sigma_{t-1}^2 = \sigma_u^2$. Note, when using this approximation to compare models with different numbers of parameters we should always use the same m.

Additionally, the second derivative matrix of $-logL$ (at the MLE) is the observed *information matrix*, whose inverse is an approximation to the variance-covariance matrix of the estimators.

Splus - ARMA Parameter Estimation

The standard Splus function `arima.mle(x, model, ...)` allows to estimate the parameters of ARMA models.

The *required arguments* to call `arima.mle(x, model)` are `x`, the univariate time series or vector. Missing values (NAs) are allowed, and `model`, the list of parameters specifying the ARIMA model. If a simple ARIMA model is specified (i.e., only one component), then model should be a list with component names from: `order`, `period`, `ar`, `ma`, `ndiff`, `ar.opt`, `ma.trans`, and `ma.opt`.

Optional arguments include `n.cond`, the number of observations on which to condition the likelihood. `n.cond` must be at least P+D (the default), where P and D are the orders of the expanded autoregressive and differencing polynomials. `xreg` is a univariate or multivariate time series or a vector or matrix with univariate time series per column. These will be used as additive regression variables. If `xreg` is of length `1`, the mean of the time series will be estimated. Through `...` additional arguments can be passed to `nlmin()`, a general quasi-Newton optimizer under Splus, to find the minimum of the log likelihood function.

The *return value* is a list representing the result of the fitted model: `model`, the same as the input model with the estimated coefficients substituted in the components `ar` and `ma`. `var.coef`, the variance-covariance matrix for the autoregressive and moving average coefficients. `loglik`, -2 times the log likelihood of the model (up to a constant factor). `aic`, Akaike's information criteria, which is `loglik` plus 2 times the number of parameters fit. `sigma2`, the estimated innovations variance. `n.used`, the number of observations used to compute the likelihood. `n.cond`, the number of observations on which the likelihood is conditioned. `reg.coef`, the estimated regression coefficients (returned if `xreg` is provided). `converged`, if the optimizer has apparently successfully converged to a minimum, then `converged` is TRUE; otherwise converged is FALSE. `conv.type`, a character string describing the type of convergence. `series`, a character string giving the name of `x`, including transformations. `reg.series`, a character string giving the name of `xreg`, including transformations (only returned if `xreg` is provided).

Example: ARMA Modelling, Parameter Estimation - `xmpArimaEstimation`:

Apply the standard Splus function `arima.mle(x, model, ...)` to estimate the parameters for the simulated subset AR model specified in the previous example. We use the subset AR(5) model favored by the PACF plot:

```
# Estimate parameters:
  x.mle <- \textsc{Arima}.mle(x-mean(x),
      model=list(order=c(5, 0, 0), ar.opt=c(T, T, F, F, T)), n.cond=10)
# Print AR coefficients and errors:
  ar <- x.mle$model$ar
  std <- sqrt(diag(x.mle$var.coef))
  data.frame(cbind(ar,std))
```

The result is:

```
          ar      std
1 -0.389363 0.031624
2  0.098744 0.033959
3  0.000000 0.034104
4  0.000000 0.033959
5  0.099484 0.031624
```

Model Selection

We want to find a model that fits the observed data as good as possible. Initial model identification is done using the ACF and PACF as briefly explained before. An additional procedure

for selecting the model order is the use of a penalized log-likelihood measure, for example the Akaike's Information Criterion (AIC)

$$AIC(p,q) = -2l(\Theta|X) + \frac{2(p+q)}{T} = log\widehat{\sigma}^2 + \frac{2(p+q)}{T} \ , \tag{2.26}$$

with the log-likelihood function $l(\Theta|X)$, or for example Schwarz's Criterion (BIC) that has in some sense better statistical properties

$$BIC(p,q) = -2l(\Theta|X) + \frac{(p+q)}{TlogT} = log\widehat{\sigma}^2 + \frac{(p+q)}{TlogT} \ . \tag{2.27}$$

There exist a variety of other selection criteria that may be used to choose an appropriate model. All these models have in common to be structured in terms of the estimated error variance $\hat{\sigma}^2$ plus a penalty adjustment involving the number of estimated parameters. It is in the extent of this penalty that the criteria differ. The criteria are used in the way that with upper bounds, say p_{max} and q_{max} orders p and q are selected such that the values given by the criterions are becoming a minimum.

Example: ARMA Modelling, Model Selection - `xmpArimaSelection`:

Now let us estimate the parameters for all models with $p_{max} = 6$ and $q_{max} = 6$ for the simulated time series from the previous example. Let us see if the subset model will be the favored model by the AIC criterion if we also include this model into consideration.

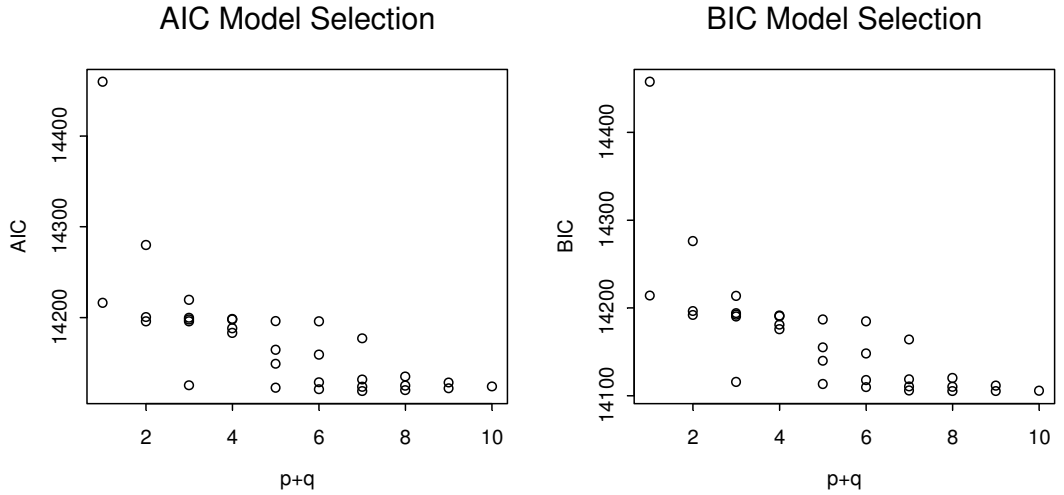
Execute the following Splus functions:

```
# Simulate a Subset AR(5) process ...
  n <- 5000
  x <- \textsc{Arima}.sim(n=n, n.start=100, rand.gen=rnorm,
      model=list(order=c(5,0,0), ar.opt=c(T,T,F,F,T), ar=c(-0.4,0.1,0,0,0.1)))
# Perform Maximum Loglikelihhood Estimation ...
  x <- x - mean(x)
  ncond <- 10
  mle <- \textsc{Arima}.mle(x,
      model=list(order=c(5,0,0), ar.opt=c(T,T,F,F,T), n.cond=ncond))
  p <- 3;q <- 0; r <- 3
  aic <- mle$aic
  bic <- mle$loglik + r/log(n-ncond)
  for (i in 0:8) {
      for (j in 0:4) {
          if (i+j > 0) {
              p <- c(p, i); q <- c(q, j); r <- c(r, i+j)
              mle <- \textsc{Arima}.mle(x, model=list(order=c(i, 0, j), n.cond=ncond))
              aic <- c(aic, mle$aic)
              bic <- c(bic, mle$loglik + (i + j)/log(n-ncond) ) }
      }
  }
  data.frame(cbind(p, q, r, aic, bic))
  plot(c(r), c(aic))
  plot(c(r), c(bic))
```

	p	q	r	aic	bic		p	q	r	aic	bic		p	q	r	aic	bic
1	3	0	3	14125	14116	2	0	1	1	14460	14458	3	0	2	2	14280	14276
4	0	3	3	14219	14214	5	1	0	1	14216	14214	6	1	1	2	14200	14196
7	1	2	3	14200	14194	8	1	3	4	14188	14181	9	2	0	2	14196	14192

```
10  2  1  3  14198 14192   11  2  2  4  14198 14191   12  2  3  5  14164 14155
13  3  0  3  14196 14190   14  3  1  4  14198 14190   15  3  2  5  14196 14187
16  3  3  6  14159 14148   17  4  0  4  14183 14176   18  4  1  5  14149 14140
19  4  2  6  14196 14184   20  4  3  7  14177 14164   21  5  0  5  14122 14113
22  5  1  6  14129 14117   23  5  2  7  14132 14119   24  5  3  8  14135 14120
25  6  0  6  14121 14110   26  6  1  7  14123 14110   27  6  2  8  14125 14110
28  6  3  9  14128 14111   29  7  0  7  14119 14106   30  7  1  8  14120 14105
31  7  2  9  14122 14105   32  7  3 10  14124 14105
```

The result is shown in figure 2.1.5:



■ Figure 2.1.5 displays to the left the AIC and to the right the BIC statistics for data simulated from a subset AR(5) time series model with parameters $\phi_1 = -0.4$, $\phi_2 = \phi_5 = 0.1$. Considering the fully specified models the favored model is the AR(5) model (the lowest circle at $p + q = 5$). However, including the subset AR(5) one finds that this model becomes clearly favored (the lowest circle at $p + q = 5$).

Diagnosis Checking

The residuals are estimated by subtraction of the adopted model from the observed time series, for an ARMA(p,q) process:

$$\widehat{\mu}_t = x_t - \phi_1 x_{t-1} - ... - \phi_p x_{t-p} + \theta_1 \widehat{\mu}_{t-1} + ... + \theta_q \widehat{\mu}_{t-q}. \tag{2.28}$$

If the process is really "true" we expect the residuals to be *iid*.

Splus - ARMA Diagnosis Checking

The standard Splus function `arima.diag(z, ...)` computes diagnostics for an ARIMA model. The diagnostics include the autocorrelation function of the residuals, the standardized residuals, and the portmanteau goodness of fit test statistic.

The *required argument* `z` is a list like output from `arima.mle()`.

The *optional arguments* include: `acf.resid`, a logical flag, if TRUE, the autocorrelation of the residuals will be returned. `lag.max`, the maximum number of lags at which to estimate the autocovariance. If this is not supplied, it is the maximum between gof.lag plus the number of model parameters and a number proportional to the logarithm of the length of the series. `gof.lag`, if positive, then `gof.lag` plus the number of model parameters is the number of lags to use for computing the Portmanteau goodness of fit statistic. If zero, then the statistic will not be computed. `resid`, a logical flag:, if TRUE, then the residuals will be returned. `std.resid`, a logical flag: if TRUE, then the standardized residuals will be returned. `plot`, a logical flag: if TRUE, the diagnostics will be plotted using the function `arima.diag.plot()`. `...` additional arguments may be passed to the function `arima.diag.plot()`.
The *returned value* consists of a list (which is returned invisibly when plot=TRUE) with the following elements: `acf.list`, a list representing the autocorrelation function of the residuals. `gof`, a list representing the Portmanteau goodness of fit statistics computed for a range of lags. The list has four elements: `lag`, `statistic`, `df`, `p.value`. `lag` is a vector of the number of lags used to compute the statistics. `statistic` is the vector of statistics corresponding to each lag used. `df` is the number of degrees of freedom the test statistics have under the null hypothesis that the model is correct. `p.value` is a vector of the p-values for the statistics using a Chi-Squared distribution with the appropriate degrees of freedom. `resid`, the residuals or innovations for the process. `std.resid`, the standardized residuals. The residuals are standardized to have unit variance under the assumption that the model is correct and the process is Gaussian. `series`, the name of x, including transformations.

Example: ARMA Modelling, Diagnosis Checking - `xmpArimaDiagnostics:`

A quick diagnosis check looks like follows:

```
# Simulate Time Series:
    model <- list(order=c(5, 0, 0),
        ar.opt=c(T, T, F, F, T), ar=c(-0.40, 0.1, 0, 0, 0.1))
    x <- arima.sim(n=1000, model=model, n.start=100)
# Estimate Parameters:
    x.mle <- arima.mle(x-mean(x), model)
# Compute Diagnostics:
    arima.diag(x.mle, plot=T)
```

The graphical output is displayed in figure 2.1.6

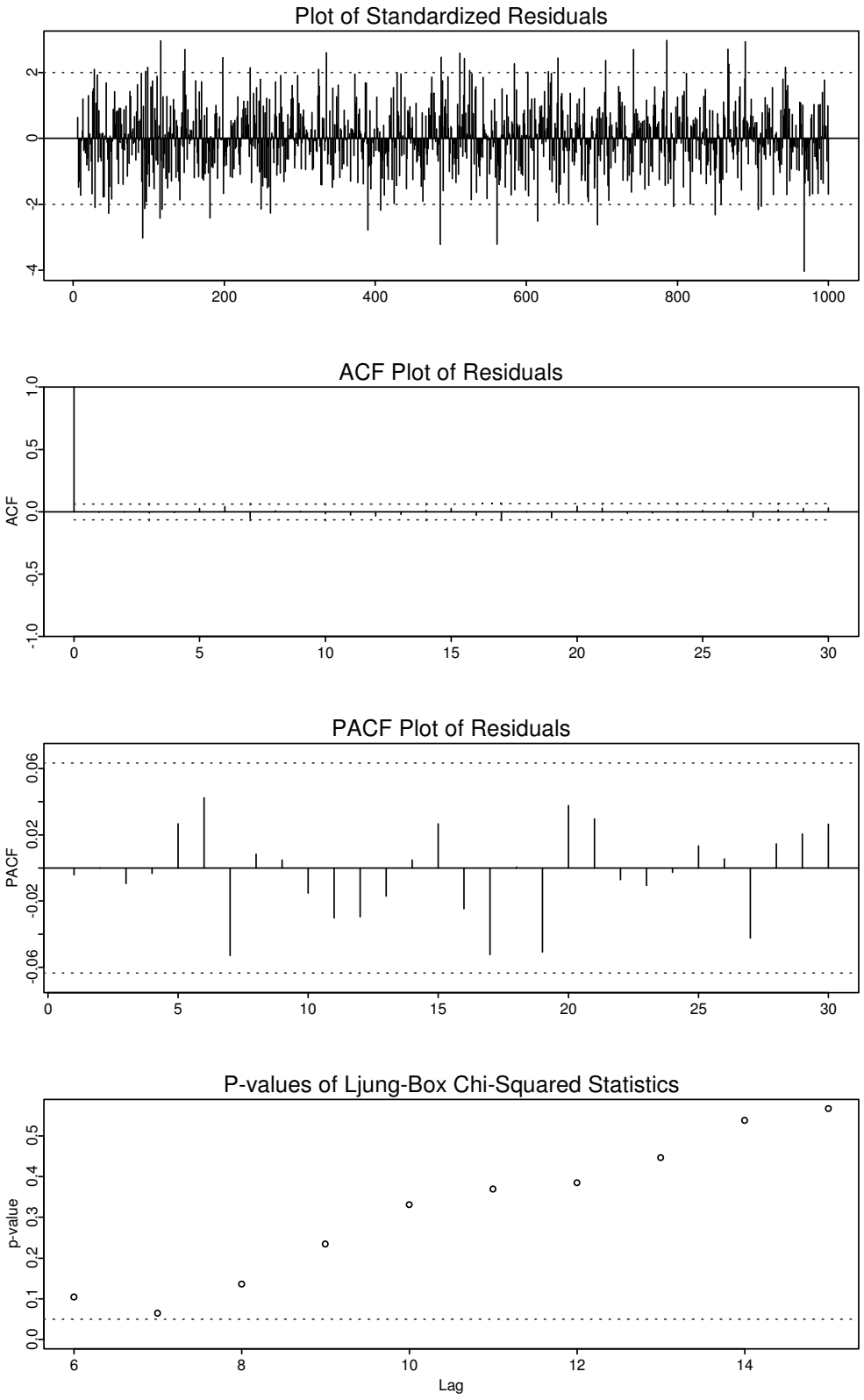
2.1.3 Forecasting Stationary Processes

Given a stochastic process up to time T, $x_1, x_2, ..., x_T$ a prominent issue within time series analysis is to provide estimates of *future variables* x_{T+h}, $h = 1, 2, ...$, conditional on the available information, i.e. $x_T, x_{T-1}, x_{T-2}, ...$ Often the information on x_t up to time T is summarized in an information set $\Omega_T = x_T,\ x_{T-1},\ x_{T-2}, ...$. Within the class of stationary ARMA(p,q) processes x_{T+h} is given as:

$$\begin{aligned} x_{T+h} &= \nu + \phi_1 x_{T+h-1} + \ phi_2 x_{T+h-2} + \ldots + \phi_p x_{T+h-p} \\ &+ u_{T+h} - \theta_1 u_{T+h-1} - \theta_2 u_{T+h-2} - \ldots - \theta_q u_{T+h-q} \ . \end{aligned} \tag{2.29}$$

A convenient forecast for x_{T+h} is the expectation of x_{T+h} given Ω_T, i.e.

$$x_{T,h} = E[x_{T+h}|x_T, x_{T-1}, x_{T-2}, \ldots] \ . \tag{2.30}$$



■ Figure 2.1.6 displays the graphical results from from the Splus function. `arima.diag()`. From top to bottom, the first graph displays the time series, the second the ACF, the third the PACF and the forth the p-values of the Ljung-Box Chi-Squared Statistics.

It can be shown that for linear time series processes the conditional mean yields forecasts which are characterized by minimum mean squared errors, $E[a_{T,h}^2] = E[(x_{T+h} - x_{T,h})^2]$. The computation of $x_{T,h}$ follows a recursive scheme of substituting

$$E[x_{T+j}|x_T, x_{T-1}, ...] = \left\{ \begin{array}{rl} x_{T+j} \ , & j \leq 0 \ , \\ E[x_{T+j}|x_T, x_{T-1}, ...] \ , & j > 0 \ , \end{array} \right. \tag{2.31}$$

and

$$E[u_{T+j}|x_T, x_{T-1}, ...] = \left\{ \begin{array}{rl} u_{T+j} \ , & j \leq 0, \\ 0 \ , & j > 0 \ , \end{array} \right\} \tag{2.32}$$

in (2.29). As an illustrative example consider the ARMA(1,1)-model in (2.12) augmented with a deterministic component. It is assumed that the parameters of the model are known:

$$x_{T+h} = \nu + \phi x_{T+h-1} + u_{T+h} - \theta u_{T+h-1} \ . \tag{2.33}$$

One obtains successively:

$$\begin{array}{rcl} x_{T,1} & = & \nu + \phi x_T - \theta u_T, \\ x_{T,2} & = & \nu + \phi x_{T,1} \\ & = & \nu + \phi(\nu + \phi x_T - \theta u_T) \ , \\ x_{T,3} & = & \nu + \phi x_{T,2} \\ & = & \nu + \phi(\nu + \phi(\nu + \phi x_T - \theta u_T)) \ . \end{array} \tag{2.34}$$

Iterating (2.34) shows that with increasing forecast horizon h the forecast $x_{T,h}$ converges to the unconditional mean of the process, i.e.

$$\lim_{h \to \infty} x_{T,h} = E[x_t] = \nu(1 + \phi + \phi^2 + \phi^3 + \ldots) = \mu \ . \tag{2.35}$$

The sequence of forecast errors $a_{T,h} = x_{T+h} - x_{T,h}$ is immediately obtained from (2.34). E.g. for $h = 1$ one has

$$\begin{array}{rcl} a_{T,1} & = & x_{T+1} - x_{T,1} \\ & = & \nu + \phi x_T + u_{T+1} - \theta u_T - (\nu + \phi x_T - \theta u_T) \\ & = & u_{T+1} \ . \end{array} \tag{2.36}$$

Pursuing along similar lines for $h = 2, 3, \ldots$ yields

$$a_{T,2} = u_{T+2} + (\phi - \theta)u_{T+1}, \tag{2.37}$$

$$a_{T,3} = u_{T+3} + (\phi - \theta)u_{T+2} + \phi(\phi - \theta)u_{T+1}. \tag{2.38}$$

Obviously all forecasts provided in (2.34) are unbiased, i.e. the expected value of the forecast errors in (2.36) to (2.37) and so on are zero. The variance of the forecast error, however, increases with h. With increasing forecast horizons h the forecast error variance converges to the unconditional variance of the process:

$$\lim_{h\to\infty} E[a_{T,h}^2] = \gamma_0 \ . \tag{2.39}$$

Splus - ARMA Forecasting

The standard Splus function `arima.forecast(x, model, ...)` forecasts a univariate time series using an ARIMA model. Under the assumption that the model is known, predicted values and their standard errors are computed for future values.

The *required arguments* include `x` the univariate time series or a vector. Missing values (NAs) are allowed. `model` is a list specifying an ARIMA model as for the function `arima.mle()`. Note that the coefficients must be provided through the elements `ar` and `ma` (otherwise the coefficients are set to zero).

Optional arguments are the following: `n`, the number of time periods to forecast beyond the end of the series (optional if end is provided). `end`, the last date for which forecasts should be produced. Forecasts will be produced for every time period between `end(x)` and `end`. `sigma2`, the estimated innovations variance. If omitted, `sigma2` will be the concentrated prediction error variance computed from the model. `xreg`, a univariate or multivariate time series or a vector or matrix with univariate time series per column. These will be used as additive regression variables. `xreg` must be consistent with the number of time periods forecast: e.g., if `n=10` and `length(x)=20`, then the number of rows of `xreg` should be 30. `reg.coef`, a vector of regression coefficients corresponding to `xreg`.

The *returned value* is a list with the following elements: `mean`, the estimated mean of the forecasts. `std.err` the approximate forecast error. In a Gaussian series with known ARIMA model structure, then each row of the matrix tsmatrix(mean-1.96*std.err, mean+1.96*std.err) is a 95% (non-simultaneous) confidence interval for the forecast in that time period. Note that `std.err` does not take into account the variability due to estimation of the ARIMA model.

Exercise: ARMA Modelling, Forecasting

Forecast the subset AR(5) model from the previous examples on the last 10 data points.

2.1.4 Case Study: ARMA Modelling of the NYSE Composite Index

ARMA modelling is mostly used to separate the linear part from the nonlinear part of economic and financial time series. Experience shows, that for logarithmic returns an ARMA analysis usually provides models of low orders in p and q. An AR(1), AR(2) supported by the PACF plot, or ARMA(1,1), ARMA(2,1), ARMA(1,2) are the outcomes from the statistical investigation. And also information criterion statistics like AIC and BIC exhibit for these numbers of parameters their minimal values.

In this case study we like to investigate the "Composite Index" of the New York Stock Exchange (NYSE). This capitalization-weighted index measures the changes in aggregate market value of all the NYSE common stocks, more than 3000. The Index is frequently adjusted to eliminate the influence of corporate actions (mergers and spinoffs) to ensure that it reflects only movements resulting from actual stock price changes.

Exercise: ARMA Modelling of the NYSE Composite Index

Investigate the partial autocorrelation function for the NYSE Composite Index with the help of the S-plus function `acf()`.

Calculate with the help of the Splus function `arima.mle()` the AIC and BIC information criterion statistics for low dimensional AR, MA and ARMA models. What are the best models?

Calculate the residuals and the summed residual error for an AR(1), AR(2) and a subset AR(5) model with coefficients $p = 3$ and $p = 4$ kept to zero.

Investigate the residual for an AR(2) process in more detail. Investigate density and dependencies: QQ-plot, runs test, and correlation tests.

Notes and Comments

ARIMA modelling is the standard approach for linear time series analysis. The classical text book, cited in almost every paper on time series analysis is *Time Series Analysis: Forecasting and Control* written by Box and Jenkins (1976). In this section we followed the ARIMA modelling approach as presented in the modern textbook *The Econometric Modelling of Financial Time Series* written by Mills (1999).

Today, every statistical software package offers functions for the simulation, parameter estimation, diagnosis checking and forecasting in the framework of the ARIMA modelling approach. Under Splus these functions are `arima.sim()`, `arima.mle()`, `arima.diag()`, `arima.forecast()`, and `arima.filter()`. A somewhat limited functionality is also available in the `ts` package under R: `Arima0()` is a preliminary version, and is announced to be replaced in due course. The standard errors of prediction exclude the uncertainty in the estimation of the ARMA model and the regression coefficients. The results are likely to be different from Splus' `arima.mle()`, which computes a conditional likelihood and does not include a mean in the model. Alternatively Trapletti's R package `tseries` also includes functions to oerform an ARMA analysis. In addition it is worth to note that the convention used by Splus' `arima.mle()` reverses the signs of the MA coefficients. Additional useful Splus functions for time series analysis are part of the SplusTS package written by Meeker (1998). From his functions we implemented `arima.roots()`, `acf.true()`, `arima.iden()` and `arima.esti()` into our `fSeries` library.

There exist many extensions to the ARIMA modelling approach presented in this Section. This includes non-Gaussian innovations, vector VARIMA processes, seasonal SARIMA processes, and fractionally integrated ARFIMA processes. Fur a further study we refer to the cited references.

2.2 GARCH Modelling: Mastering Heteroskedastic Processes

A major contribution of the ARCH literature is the finding that apparent changes in the volatility of economic time series may be predictable and result from a specific type of nonlinear dependence rather than exogenous structural changes in variables.

Bera and Higgins 1993

Introduction

A dominant feature of the return series is volatility clustering. The conditional variance of ε_t appears to be large if recent observations $\varepsilon_{t-1},\ \varepsilon_{t-2}, ...$ are large in absolute value and smaller conditional variances are observed during periods where lagged innovations are also small in absolute value. This effect cannot be explained by linear models such as ARMA and difficult to produce in classical regression models on "returns" based predictors. This fact led to the introduction of a new kind of nonlinear models, called ARCH, Engle (1982), and GARCH, Bollerslev (1986).

2.2.1 Autoregressive Conditional Heteroskedastic Processes: ARCH(p)

The ARCH model is set up such that the conditional variances of the process depend on their past values:

$$\begin{aligned} u_t|\Omega_{t-1} &\sim \mathcal{N}(0,\sigma_t^2),\ t=1,2,... \ , \\ \sigma_t^2 &= \omega+\alpha_1 u_{t-1}^2+\alpha_2 u_{t-2}^2+...+\alpha_q u_{t-q}^2 \ . \end{aligned} \tag{2.40}$$

An intrinsic property of the definition in (2.40) is that the second order moment of u_t is given conditional on an information set Ω_t containing especially the history of the process. The unconditional residuals u_t will then be leptokurtic. The conditional and unconditional mean of the process are zero and the autocorrelation function of the residuals u_t vanishes for lags $\tau > 0$. Under suitable properties of ω and $\alpha_i,\ i=1,...,q$, lagged innovations which are large in absolute value provide a conditional variance of u_t which is also large. u_t is a heteroskedastic time series or error sequence. The process in (2.40) is termed ARCH process since heteroskedasticity is parameterized conditionally in an autoregressive manner.

Complementary to this definition one has to impose convenient conditions ensuring the volatility (conditional variance) process to be positive. Sufficient conditions that guarantee $\sigma_t^2 > 0$ are easily seen to be:

$$\omega > 0, \quad \alpha_i \geq 0,\ i=1,...,q \ . \tag{2.41}$$

Note that at least one parameter α_j should be larger than zero to yield time dependent variances. The parameter q determines the maximum order of lagged innovations which are supposed to

have an impact on current volatility. Another representation of the ARCH process immediately reveals that the process u_t fails to be linear. Let ε_t denote an independent and identically distributed error sequence with mean zero and a variance which is equal to unity. Thus the process in (2.40) can also be written as:

$$\begin{aligned} u_t &= \varepsilon_t \sigma_t \ , \\ &= \varepsilon_t \sqrt{\sigma_t^2} \ , \\ &= \varepsilon_t \sqrt{\omega + \alpha_1 u_{t-1}^2 + \alpha_2 u_{t-2}^2 + ... + \alpha_q u_{t-q}^2} \ . \end{aligned} \tag{2.42}$$

Assuming e.g. $q = 1$ is obvious that u_t is related to ϵ_{t-1} in a nonlinear fashion. As given above the conditional distribution of u_t is not further specified. To make Maximum-Likelihood or Quasi-Maximum Likelihood estimation feasible it is often assumed that u_t is conditionally normally distributed or equivalently that ε_t follows a standard normal distribution:

$$u_t \sim \mathcal{N}(0, \sigma_t^2) \ , \quad \varepsilon_t \sim \mathcal{N}(0,1) \ . \tag{2.43}$$

For the return series provided in the section about ARMA modelling it was not that clear whether these series exhibit a significant linear dependence. A convenient process which is characterized by linear and higher order dependence simultaneously is for example an autoregressive process of order p generated from an ARCH(q)-error sequence:

$$\begin{aligned} x_t &= \nu + \phi_1 x_{t-1} + ... + \phi_p x_{t-p} + u_t \ , \\ u_t | \Omega_{t-1} &\sim \mathcal{N}(0, \sigma_t^2) \ , \\ \sigma_t^2 &= \omega + \alpha_1 u_{t-1}^2 + \alpha_2 u_{t-2}^2 + ... + \alpha_q u_{t-q}^2 \ . \end{aligned} \tag{2.44}$$

Along similar lines one may also replace homoskedastic error terms in standard regression models by conditionally heteroskedastic error sequences. The variance process in (2.40) is specified as a function in lagged squared variables u_{t-i}^2. Imposing the aforementioned positivity constraints on ω and α_i one obtains a process showing marked clusters of volatility. To achieve this property, however, it is not necessary to specify the volatility function in terms of lagged squared variables u_{t-i}^2. In principle any function of u_{t-i} which is monotonically increasing may serve to generate volatility clusters. As a prominent example one may regard e.g. the absolute value of lagged variables yielding an alternative parameterization of the conditional variance:

$$\sigma_t = \omega + \tilde{\alpha}_1 |u_{t-1}| + \tilde{\alpha}_2 |u_{t-2}| + ... + \tilde{\alpha}_q |u_{t-q}| \ . \tag{2.45}$$

Since σ_t^2 is the conditional expectation of u_t^2 , however, a specification as given in (2.40) appears to be more natural. In addition, a specification like (2.40) facilitates the derivation of some properties of u_t in practice. Volatility clusters may cover a long period of successive observations. For practical purposes it might become necessary to choose a conveniently high order q to allow long range dependence of current volatility on lagged innovations. From a time series analysts point of view it would be appealing to have a more parsimonious class of conditionally heteroskedastic processes capturing the feature of long lasting volatility clusters.

2.2.2 Generalized Autoregressive Conditional Heteroskedasticy: GARCH(p,q)

The generalization of stationary pure autoregressive processes towards the class of ARMA models allowed to capture relatively complicated correlation structures without violating the principle of parsimonious time series modelling. An analogous reasoning can be made for the provision of the generalized ARCH model, i.e. the GARCH(q,p)-process. This conditionally heteroskedastic process is obtained by augmenting (2.40) with a component which is autoregressive in σ_t^2 :

$$\begin{aligned} u_t|\Omega_{t-1} &\sim (0, \sigma_t^2),\ t = 1, 2, ..., \\ \sigma_t^2 &= \omega + \alpha_1 u_{t-1}^2 + \alpha_2 u_{t-2}^2 + ... + \alpha_q u_{t-q}^2 + \beta_1 \sigma_{t-1}^2 + ... + \beta_p \sigma_{t-p}^2 \ . \end{aligned} \tag{2.46}$$

Sufficient conditions for the conditional variances to be positive are obviously:

$$\omega > 0, \quad \alpha_i, \beta_j \geq 0,\ i = 1, ..., q,\ j = 1, ..., p \ . \tag{2.47}$$

Using lag-polynomials the specification of σ_t^2 in (2.46) may also be given as:

$$\begin{aligned} (1 - \beta_1 B - ... - \beta_p B^p)\sigma_t^2 &= \omega + (\alpha_1 B + ... + \alpha_q B^q)u_t \ , \\ (1 - \beta(B))\sigma_t^2 &= \omega + \alpha(B)u_t \ . \end{aligned} \tag{2.48}$$

Assuming the roots of the polynomial $(1 - \beta(z))$ to be larger than one in absolute value the model can also be written as an ARCH process of infinite order:

$$\sigma_t^2 = (1 - \beta(B))^{-1}\omega + (1 - \beta(B))^{-1}\alpha(B)u_t \ . \tag{2.49}$$

Defining an uncorrelated series with zero mean as

$$\begin{aligned} v_t &= u_t^2 - \sigma_t^2 \ , \\ \Leftrightarrow \sigma_t^2 &= u_t^2 - v_t \ , \end{aligned} \tag{2.50}$$

the GARCH(q,p) yields a representation of squared errors ε_t by substitution into (2.46) which is similar to an ARMA process:

$$\begin{aligned} u_t^2 &= \omega + \alpha_1 u_{t-1}^2 + ... + \alpha_q u_{t-q}^2 + v_t + \beta_1(u_{t-1}^2 - v_{t-1}) + ... + \beta_1(u_{t-p}^2 - v_{t-p}) \\ &= \omega + \sum_{i=1}^{max_{p,q}} (\alpha_i + \beta_i)u_{t-i}^2 + v_t - \sum_{i=1}^{p} \beta_i v_{t-i}. \end{aligned} \tag{2.51}$$

Although the error term v_t can be shown to be uncorrelated the process defined in (2.50) is still dependent with respect to higher order moments mitigating to some extend the interpretation of (2.51) as an ARMA$(max_{p,q}, p)$-process for squared variables ε_t^2 . Since v_t is uncorrelated,

however, it becomes possible to compute directly the autocorrelation pattern of ε_t^2 implied by a GARCH(q,p) process from its ARMA($max_{p,q}, p$) representation. Assume for simplicity ε_t^2 to follow a GARCH(1,1) specification. The implied ARMA representation for squared variables ε_t^2 is

$$u_t^2 = \omega + (\alpha_1 + \beta_1)u_{t-1}^2 + v_t - \beta_1 v_{t-1} \ . \tag{2.52}$$

From the discussion of the stationary ARMA(1,1) process the implied autocorrelation pattern of u_t^2 is immediately obtained by choosing

$$\phi = \alpha_1 + \beta_1, \, \theta = -\beta_1 \ . \tag{2.53}$$

In the ARMA(1,1)-model the parameter ϕ was crucial in describing how fast linear dependence dies out at increasing horizons $k = 2, 3, ...$. In (2.51) the corresponding parameter is $\alpha_1 + \beta_1$, i.e. linear dependence of squared variables ε_t^2 dies out according to the following equation:

$$\rho_k(\varepsilon^2) = (\alpha_1 + \beta_1)\rho_{k-1}(\varepsilon^2), \ k = 2, 3, ... \ . \tag{2.54}$$

The initial level of autocorrelation, i.e. $\rho_1(u^2)$ is essentially determined by $(\phi + \theta)$ in the ARMA(1,1) model and thus by $\alpha_1 + \beta_1 - \beta_1 = \alpha_1$ in the representation given above.

The ARMA($max_{p,q}, p$) representation for u_t is also useful to derive the unconditional expectation of squared GARCH-type error terms, which is for the GARCH(1,1)-process

$$E[u_t^2] = \frac{\omega}{(1 - \alpha_1 - \beta_1)} \ . \tag{2.55}$$

Note that the unconditional expectation of u_t^2 or say the unconditional variance of u_t only exists if $(\alpha_1 + \beta_1)$ is less than one. For the general GARCH(p,q) process the unconditional variance of u_t exists if

$$\beta(1) + \alpha(1) < 1 \ . \tag{2.56}$$

holds. The unconditional variance of u_t may also be evaluated using the law of iterated expectations. Since this procedure is helpful to derive further unconditional moments of a GARCH-type error sequence the following theorem is given: Let Ψ_1 and Ψ_2 denote two set of random variables satisfying $\Psi_1 \subseteq \Psi_2$. y denotes a scalar random variable. Then

$$E[y|\Psi_1] = E[E[y|\Psi_2]|\Psi_1] \ . \tag{2.57}$$

To make the law of iterated expectations applicable to GARCH-type processes Ψ_1 and Ψ_2 are interpreted as different information sets and Ψ_1 is defined to be the empty set which is used to derive unconditional expectations. Ψ_2 is conveniently replaced by Ω_{t-1}. In this case the unconditional expectation of u_t is obtained from (2.57) as:

$$E[u_t|\Psi_1] = E[u_t] = E[E[u_t|\Omega_{t-1}]] = 0 \ . \tag{2.58}$$

Turning to unconditional autocorrelations of ε_t and v_t or to the unconditional second order moment of ε_t the following results are obtained:

$$E[u_t u_{t_k}|\Psi_1] = 0,\ k > 0\ . \tag{2.59}$$

$$E[u_t^2] = \frac{\omega}{(1-\alpha_1-\beta_1)}\ . \tag{2.60}$$

$$E[v_t v_{t-k}] = 0\ . \tag{2.61}$$

In addition, under normality the following result is obtained for the unconditional fourth order moment:

$$E[u_t^4] = \frac{3\omega^2(1+\alpha_1+\beta_1)}{(1-\alpha_1-\beta_1)(1-\beta_1^2-2\alpha_1\beta_1-3\alpha_1^2)}\ . \tag{2.62}$$

Note that the unconditional fourth order moment of u_t only exists if

$$(\beta_1^2+2\alpha_1\beta_1+3\alpha_1^2) < 1\ . \tag{2.63}$$

holds. The unconditional fourth order moment is used to compute the kurtosis of the unconditional distribution of ε_t. The kurtosis is defined as

$$\kappa = \frac{E[u_t^4]}{(E[u_t^2])^2} - 3\ . \tag{2.64}$$

Normally distributed variables have a kurtosis equal to $\kappa = 0$. For the unconditional distribution of ε_t κ is obtained as:

$$\kappa = \frac{3(1-\alpha_1-\beta_1(1+\alpha_1+\beta_1)}{(1-\beta_1^2-2\alpha_1\beta_1-3\alpha_1^2)} - 3\ . \tag{2.65}$$

For $\alpha_1 = \beta_1 = 0$ the kurtosis in (2.65) reduces to the kurtosis of the normal distribution, i.e. $\kappa = 30$. Setting $\beta_1 = 0$ the implied unconditional kurtosis of the ARCH(1)-model under normality is obtained:

$$\kappa = 3\frac{(1-\alpha_1^2)}{(1-3\alpha_1^2)} - 3 > 0, \quad \text{since } 0 < \alpha_1 < 1\ . \tag{2.66}$$

2.2.3 Modelling under Conditional Heteroskedasticity

Parameter Estimation

So far knowledge of the underlying parameters, of the GARCH models was assumed. In practice, however, such knowledge is not available and given a sequence of observed time series variables the underlying parameters are to be estimated. The maximum likelihood approach requires iterative optimization routines. For such processes one has to have a reasonable choice of initial parameters starting the estimation procedure.

To provide a general framework for the issue of estimating GARCH-type models the regression model with conditionally heteroskedastic error terms is investigated. The model is given as:

$$x_t = w_t b + u_t, \quad u_t|\Omega_{t-1} \sim \mathcal{N}(0, \sigma_t^2) \ . \tag{2.67}$$

In (2.67) w_t contains a set of explanatory variables which determines the conditional mean of x_t, i.e. $E[x_t|w_t] = w_t b$. In principle w_t may contain fixed regressor variables or even stochastic terms, e.g. lagged dependent variables or lagged error terms or even both. ε_t is an error term the distribution of which is specified conditional on an information set Ω_{t-1}. The conditional distribution is assumed to be Gaussian in order to make maximum likelihood estimation of the unknown parameters in (2.67) feasible. If the normality assumption in (2.67) is violated and thus the model given above is misspecified the estimation technique detailed below is known as quasi maximum likelihood estimation. It is assumed that the error terms in (2.67) follow a GARCH(q,p)-process, such that the following compact notation is introduced:

$$\begin{aligned} \sigma_t^2 &= \omega + \alpha_1 u_{t-1}^2 + ... + \alpha_q u_{t-q}^2 + \beta_1 \sigma_{t-1}^2 + ... + \beta_p \sigma_{t-p}^2 \\ &= \omega + \alpha_1 (x_{t-1} - w_{t-1} b)^2 + ... + \alpha_q (x_{t-q} - w_{t-q} b)^2 + \beta_1 \sigma_{t-1}^2 + ... + \beta_p \sigma_{t-p}^2 \ . \end{aligned} \tag{2.68}$$

The maximum likelihood estimator is that specific parameter vector θ that maximizes the likelihood function or equivalently the log-likelihood function. The latter is denoted as follows:

$$l(\theta|x_t, w_t) = \sum_{t=1}^{T} l_t \tag{2.69}$$

$$= \sum_{t=1}^{T} (-\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\sigma_t^2) - \frac{1}{2}\frac{\varepsilon_t^2}{\sigma_t^2})$$

$$= \sum_{t=1}^{T} (-\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\sigma_t^2) - \frac{1}{2}\frac{(x_t - w_t b)^2}{\sigma_t^2}) \ . \tag{2.70}$$

Compared to the common linear regression model with independent error terms the maximum of the likelihood function cannot be obtained analytically. However, one may run iterative maximization routines which evaluate the log-likelihood function numerically. Assuming some regularity conditions, e.g.

$$u_t = \xi_t \sigma_t,\ E[\xi_t] = 0,\ E[\xi_t^2] = 1,\ E[\xi_t^4] < \infty\ , \tag{2.71}$$

the vector of estimated parameters $\widehat{\pi}$ can be shown to be asymptotically distributed as follows:

$$\sqrt{T}(\widehat{\theta} - \theta) \xrightarrow{d} \mathcal{N}(0, D^{-1}SD^{-1})\ . \tag{2.72}$$

In (2.72) S is the expectation of the cross-product of first order derivatives of $l_t(\theta)$ with respect to θ and D is the negative expectation of the matrix of second order derivatives, i.e.

$$D = \frac{1}{T}\sum_{t=1}^{T} -E[\frac{\partial^2 l_t}{\partial\theta\partial\theta'}] \quad \text{and} \quad S = \frac{1}{T}\sum_{t=1}^{T} E[\frac{\partial l_t}{\partial\theta}\frac{\partial l_t}{\partial\theta'}]. \tag{2.73}$$

In case of normally distributed errors one has $D = S$ and the asymptotically valid covariance matrix is:

$$\sqrt{T}(\widehat{\theta} - \theta) \xrightarrow{d} \mathcal{N}(0, S). \tag{2.74}$$

To make inference feasible it is of course necessary to replace the unknown matrices S and D by convenient estimates

$$\widehat{D} = -\frac{1}{T}\sum_{t=1} T\frac{\partial^2 l_t}{\partial\theta\partial\theta'}|_{\theta=\widehat{\theta}} \quad \text{and} \quad \widehat{S} = \frac{1}{T}\sum_{t=1} T\frac{\partial l_t}{\partial\theta}\frac{\partial l_t}{\partial\theta'}|_{\pi=\widehat{\theta}}. \tag{2.75}$$

Estimation of D is only necessary if the estimated innovations are not normally distributed.

Forecasting

A common task in time series analysis is to provide forecasts of $u_{T,h}$ given information up to time T. As mentioned above for the GARCH(1,1) process $u_{T,h} = 0$ for all T, h. Often a point forecast is provided jointly with a confidence interval covering the true value u_{T+h} with probability $(1 - \alpha)$. To provide such a confidence interval one has to estimate the variance of u_{T+h} conditional on Ω_T. Forecasts of the conditional variance are most easily obtained from its ARMA representation which is given for the GARCH(1,1) model as:

$$\sigma_t^2 = \omega + (\alpha_1 + \beta_1)\sigma_{t-1}^2 + \alpha_1 v_{t-1}\ . \tag{2.76}$$

Forecasts of σ_{T+h} conditional on Ω_T can be computed recursively. If $\alpha_1 + \beta_1 < 1$ holds these forecasts converge to the unconditional variance implied by the process. Starting in periods where volatility is high this forecasted variances converge from above to the unconditional variance implying that forecasting intervals become smaller with increasing forecasting horizon h. Starting in periods of lower variance forecasted variances converge from below to their unconditional counterpart implying that forecasting intervals become larger with increasing h. Remember from the discussion of ARMA-type processes that forecasting intervals obtained for such processes always become larger with increasing h.

2.2.4 Asymmetric Volatility Models: APARCH(p,q)

As mentioned a prominent feature of empirical variance processes are so-called volatility clusters meaning that innovations which are large in absolute value tend to be followed by innovations which are also large and vice versa. Due to the positivity constraints necessary to guarantee the implied variances to be greater than zero a specific implication of GARCH-type processes is that the larger u_t , the larger is the implied conditional variance σ^2_{t+h} for all $h > 0$, i.e. a random oscillatory behavior of σ^2_{t+h} is ruled out by definition.

Another characteristic of GARCH processes is that the conditional variance is symmetric in lagged innovations u_{t-i}. Positive and negative innovations which are the same in absolute value imply the same conditional variance σ^2_{t+h} . Note that since $u_t = E[x_t|w_t]$ is uncorrelated with its history one may interpret u_t as a convenient measure of news appearing in the market in time t. From the empirical literature on returns of risky assets it is known, however, that future volatility is much more affected by negative news compared to positive news. Such an effect might be explained as follows: A decreasing stock price increases the firms debt/equity ratio making the firm "more risky" and thus increasing future volatility. This feature has become popular as the so-called *leverage effect*, which is obviously not captured by GARCH-type processes.

Ding, Granger and Engle (1993) also found that the empirical autocorrelations of equity return volatility, $|r_t|^c$, were strongest for c around one confirming the Taylor effect. This empirical observation motivates them to estimate the coefficient c rather than imposed it as squares as in the conditional variance. Therefore, they proposed the asymmetric power ARCH, called APARCH specification:

$$\begin{aligned} u_t|\Omega_{t-1} &= \sigma_t \xi_t, \quad \xi_t \sim \mathcal{N}(0,1) \ , \\ \sigma_t^\delta &= \omega + \sum_{i=1}^{p} \alpha_i (|u_{t-i}| - \gamma_i u_{t-i})^\delta + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^\delta \ , \\ \omega > 0, \ \delta &\geq 0, \ -1 < \gamma_i < 1 \ , \\ \alpha_i \geq 0, \ i &= 1, \ldots, p \quad \beta_j \geq 0, \ j = 1, \ldots, q \ . \end{aligned} \tag{2.77}$$

The added flexibility of the APARCH specification is such that it nests other members of the ARCH family, for example:

- *Engle's ARCH(p)*:
  $\delta = 2$ and $\gamma_i = 0$; $\beta_j = 0$.
- *Bollerslev's GARCH(p,q)*:
  $\delta = 2$ and $\gamma_i = 0$.
- *Taylor/Schwert's GARCH(p,q)*:
  $\delta = 1$ and $\gamma_i = 0$.
- *Glosten's et al. GJR(p,q)*:
  $\delta = 2$ and $0 \leq \gamma_i < 1$. This allows negatives shocks to have a stronger effect on volatility. It is also called threshold GARCH model (TGARCH) and can be write:

$$\sigma_t^\delta = \omega + \sum_{i=1}^{p} \alpha_i^\star \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 + \sum_{i=1}^{p} \gamma_i^\star S_t^- e_{t-i}^2, \tag{2.78}$$

  where

$$\alpha_i^\star = \alpha_i (1-\gamma_i)^2, \ \gamma_i^\star = 4\alpha_i\gamma_i, \ S_i^- = \begin{cases} 1, \text{ if } \varepsilon_{\mathrm{t-i}} < 0 \\ 0, \text{ otherwise} \end{cases} \tag{2.79}$$

Leptokurtic Innovations

To make maximum likelihood estimation with heteroskedastic error terms feasible we have assumed so far the standardized innovations

$$\xi_t = \frac{u_t}{\sigma_t}, \tag{2.80}$$

and the corresponding estimates to follow a standard normal distribution. In empirical practice the normality assumption is often rejected for the estimated innovations $\widehat{\xi_t}$ of a GARCH-type variance equation. As outlined maximum likelihood estimation is consistent and provides asymptotically valid distributional results for the rescaled and centered vector of estimated parameters $\sqrt{T}(\widehat{\theta} - \theta)$ under suitable conditions. If the normality assumption is violated, however, it is no longer possible e.g. to provide valid forecasting intervals for u_{t+h} given Ω_t using quantiles of the $\mathcal{N}(0,1)$-distribution. If the distribution of ξ_t is leptokurtic forecasting intervals with nominal coverage probability $(1-\alpha)$ computed under the normality assumption will tend to have an empirical coverage probability exceeding the nominal coverage. To obtain valid forecasting intervals for u_{t+h} it pays to take a leptokurtic distribution of ξ_t into account.

A candidate allowing for excess kurtosis is the t-distribution, introduced by Bollerslev (1986). A random variable Z is said to be $t(\theta, \emptyset, \nu)$-distributed if its density function is given as:

$$f(z|\theta, \emptyset, \nu) = \frac{\nu^{\nu/2}\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi}\Gamma(\frac{\nu}{2})} \frac{1}{\sqrt{\emptyset}} \left[\nu + \frac{(z-\theta)^2}{\emptyset}\right]^{-(\frac{\nu+1}{2})}. \tag{2.81}$$

The domain of the t-distribution is $-\infty < z < \infty$. θ is the expected value of Z. The so-called scaling variable $\emptyset$ and ν, the degrees of freedom parameter, are both assumed to be positive $(\nu > 0, \emptyset > 0)$. Some remarks on the properties of this distribution are made for convenience:

- $E[Z] = \theta$, if $\nu > 1$, $Var[Z] = \frac{\nu}{\nu-2}\emptyset$, if $\nu > 2$.
- The standardized random variable $Z^\star = (Z-\theta)/\sqrt{\emptyset}$ follows a Student $t(\nu)$-distribution with density function obtained from (2.81) assuming $\emptyset = 1$ and $\theta = 0$ $(Z^\star \sim t(\nu) = t(0,1,\nu))$.
- With increasing degrees of freedom the t-density in (2.81) can be closely approximated by a normal density, $\phi(z|\mu, \sigma^2)$, with corresponding mean and variance, i.e. for $\nu \to \infty$.

$$f(z|\theta, \emptyset, \nu) \approx \phi\left(z|\theta, \frac{\nu}{\nu-2}\emptyset\right) \text{ and } \mathrm{f(z|0,1,\nu)} \approx \phi(\mathrm{z|0,1}).$$

The t-distribution is more concentrated around the zero mean compared to the standard normal distribution. In addition the t-model has fatter tails than the N(0,1) distribution. Both properties are stylized facts of financial market data.

Another attractive candidate is a (truncated) α-stable distribution as defined in Part I. However, through the lack of an easy representation of the distribution function, the maximum likelihood approach becomes from the numerical point of view difficult to handle.

Example: GARCH Time Series Simulation with Different Innovations - `xmpGarchInnovSim`:

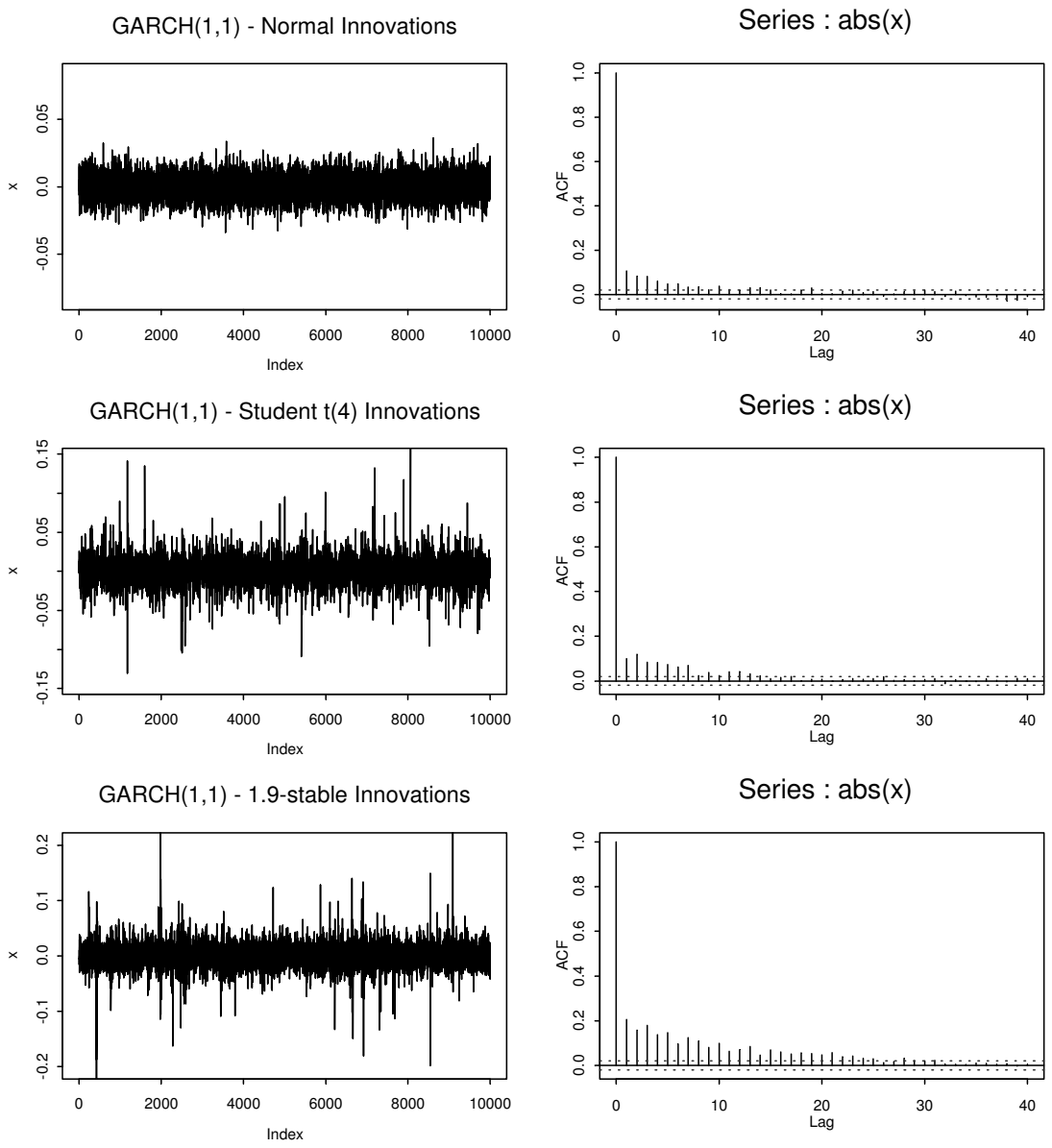
Use the Splus function `garch.sim()` from the `fSeries` library to simulate the time series of length 10000 for three symmetric Taylor-Schwert GARCH(1,1) processes with parameters $\omega = 0.001, \alpha = 0.1, \beta = 0.8, \gamma = 0, \delta = 1$ created from the following distribution functions: Normal distribution, Student t-distribution with 4 degrees of freedom, and α-stable distribution with index 1.9. Plot the result together with the associated autocorrelation functions. Although the name of the function is `garch.sim()` the function allows the simulation of the more general APARCH models allowing for innovations drawn from a general distribution.

The *arguments* for the functions are: `model`, a list including the following model parameters, `omega`, the constant coefficient of the variance equation, `alfa`, the autoregressive coefficients, `beta`, the variance coefficients (by default zero), `gamma`, the asymmetry coefficients (by default zeros), `delta`, the variance coefficient (by default 2), `alfa.selected` if not defined, an array with succeeding numbers ranging from 1 to the number of alfa coefficients q, otherwise an array with the selected lag numbers used to define in a simple way subset models, `beta.selected`, the same for the beta-coefficient. `innov`, the innovations, `start.innov`, the innovations for excluded starting values, `doplot`, a logical flag, if true the simulated time series is plotted,(by default F).

The *returned value* is the time series vector.

```
# Settings:
    set.seed(311)
    n.innov <- 10000
    model <- list(omega=1.0e-3, alfa=0.1, gamma=0,
        alfa.selected=1, beta=0.8, beta.selected=1, delta=1)

# Normal Distribution:
    innov <- rnorm(n.innov)
    start.innov <- rnorm(1000)
    x <- garch.sim (model=model, innov=innov, start.innov=start.innov)
    s <- sqrt(var(x))
    plot(x, type="l", ylim=c(-s,s)*plot.scale,
        main="GARCH(1,1) - Normal Innovations")
    acf(abs(x))

# Student t-Distribution with df=4:
    innov <- rt(n.innov, df=4)
    start.innov <- rt(1000, df=4)
    x <- garch.sim (model=model, innov=innov, start.innov=start.innov)
    s <- sqrt(var(x))
    plot(x, type="l", ylim=c(-s,s)*plot.scale,
        main="GARCH(1,1) - Student t(4) Innovations")
    acf(abs(x))

# Alpha-Stable Distribution with Index 1.9:
    innov <- rstab(n.innov, index=1.9)
    start.innov <- rstab(1000, index=1.9)
    x <- garch.sim (model=model, innov=innov, start.innov=start.innov)
    s <- sqrt(var(x))
    plot(x, type="l", ylim=c(-s,s)*plot.scale,
        main="GARCH(1,1) - 1.9-stable Innovations")
    acf(abs(x))
```

The result is shown in figure 2.2.1.



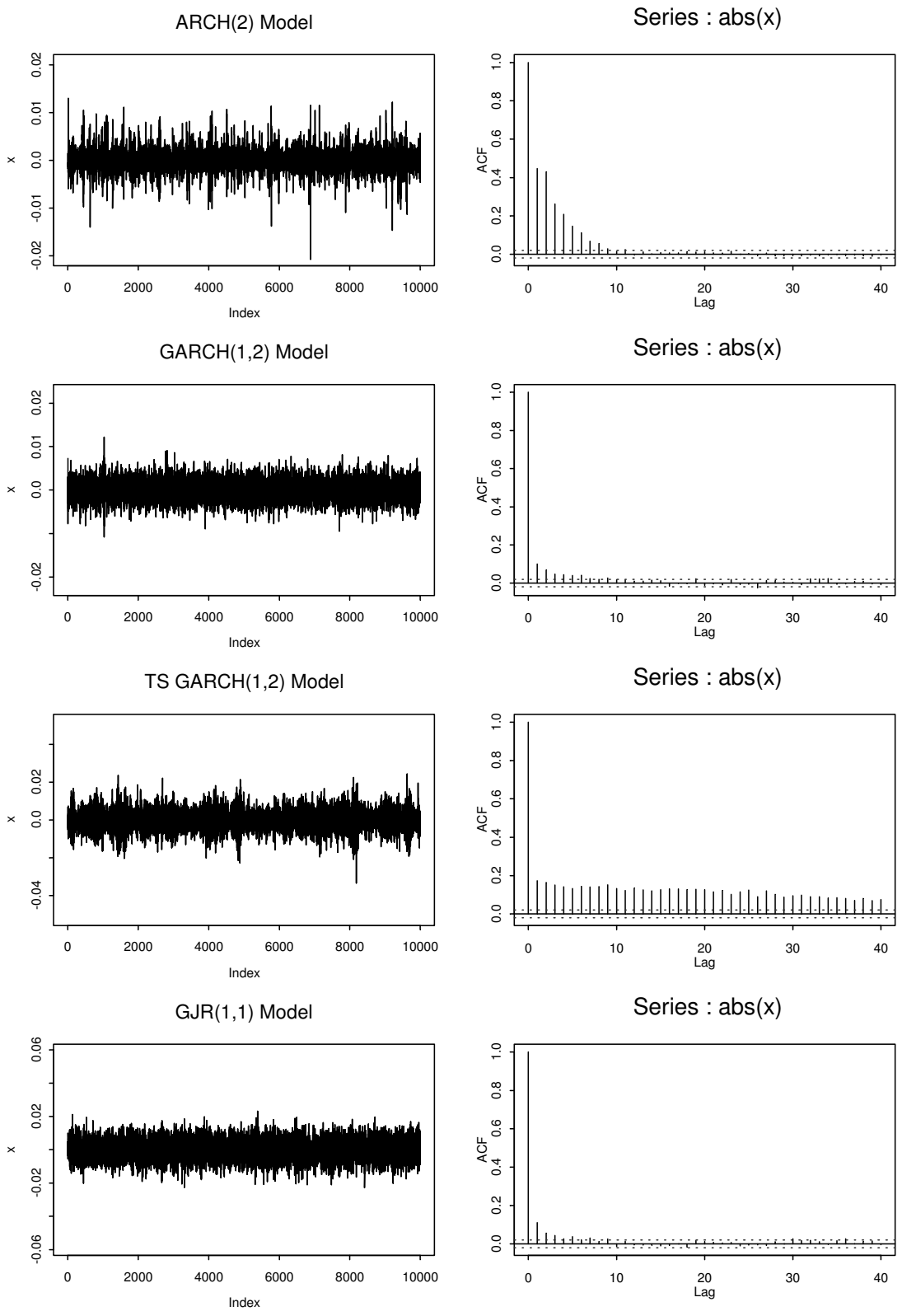
■ Figure 2.2.1 displays the time series of three simulated GARCH(1,1) processes with normal, t-distributed and α-stable innovations together with the associated autocorrelation function.

Example: APARCH Time Series Simulations - `xmpGarchAparchSim`**:**

Use the Splus function `garch.sim()` from the `fSeries` library to simulate models from the APARCH family: Engel's ARCH(p), Bollerslev's GARCH(p,q), Taylor-Schwert's GARCH(p,q), Glosten et al.'s GJR(p,q), all with normal distributed innovations, e.g.

```
# Glosten's et al. GJR(p,q) model
  x <- garch.sim (model=list(
       omega=1.0e-5, alfa=0.1, gamma=0.2, alfa.p=1, beta=0.6, beta.q=1,
       beta=0.6, delta=2), innov=rnorm(n.innov), start.innov=rnorm(1000))
```

The result is shown in figure 2.2.2.



■ Figure 2.2.2 displays the time series of four simulated APARCH(1,1) processes, from above to below: Engel's ARCH(p), Bollerslev's GARCH(p,q), Taylor-Schwert's GARCH(p,q), Glosten et al.'s GJR(p,q), all with normal distributed innovations.

Example: APARCH Parameter Estimation - `xmpGarchMle`:

Use the Splus function `garch.mle()` from the `fSeries` Library to estimate the model parameters for simulated APARCH processes including:

1 Engle's ARCH(2) with normal innovations,
2 Subset ARCH(3[1,3]) model,
3 Bollerslev's GARCH(1,1) with normal innovations,
4 Taylor-Schwert's subset TS-GARCH(1,5[1,5]) model, delta=1,
5 GARCH(1,1) normal innovations with δ-optimization,
6 GARCH(1,1) with normal innovations and γ-asymmetry,
7 GARCH(1,1) model same as from 6) and δ-optimization,
8 APARCH(1,1) with α-stable innovations with δ- and index parameter optimization.

Here follows the example for case 4:

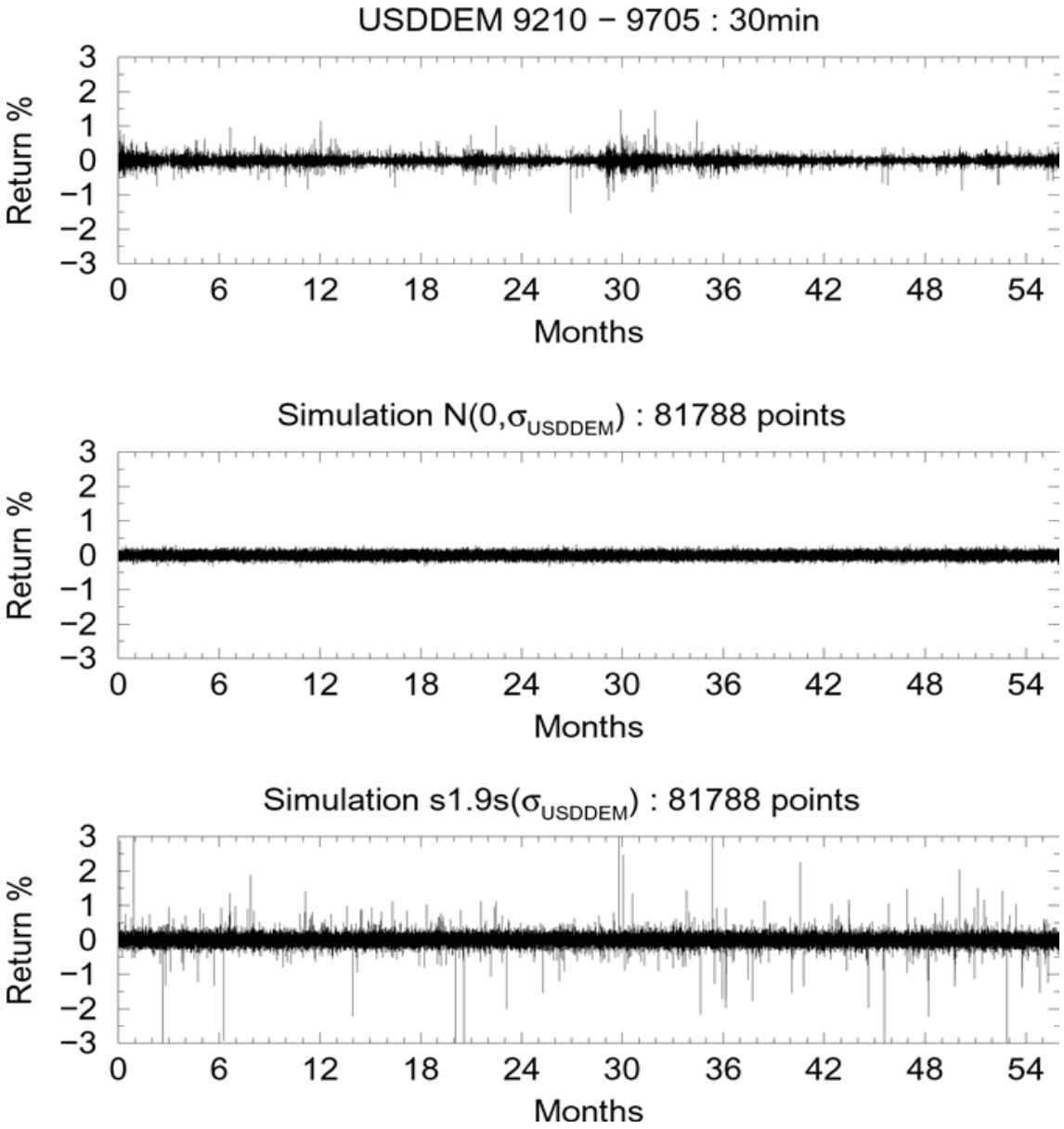
```
# 4) Taylor-Schwert's TS-GARCH(1,5[1,5]) subset model, delta=1
    xsim <- garch.sim(model=list(omega=1.0e-6, alfa=0.1,
        beta=c(0.7, 0.1), beta.selected=c(1,5), delta=1),
        innov=rnorm(n=nt), start.innov=rnorm(n=1000), doplot=T)
    parameters <- garch.mle(x=xsim, model=list(model="norm",
        omega=1.e-5, alfa=0.1, beta=c(0.2,0.2),
        beta.selected=c(1,5), delta=1), n.cond=10)
```

The result reads:

```
...
  [1] "xvec"
  [1] 1e-005 1e-001 2e-001 2e-001
  ...
STARTING PARAMETERS
  Iteration Call
  [1] 1
  Functionvalue
  [1] -98962.79
  Estimated Parameters
  [1] 1e-005 1e-001 2e-001 2e-001
ITERATION PATH
  Iteration Call
  [1] 8
  Functionvalue
  [1] -101593
  Estimated Parameters
  [1] 6.885793e-006 9.804430e-002 1.835696e-001 1.835611e-001
  ...
  Iteration Call
  [1] 326
  Functionvalue [1] -102625.3
  Estimated Parameters
  [1] 1.221429e-006 1.022095e-001 7.134607e-001 5.947400e-002
  ...
```

2.2.5 Case Study: High Freuqency USDDEM FX Rates

As an example for GARCH modelling we like to discuss here the case of high frequency US Dollar / German Mark exchange rates. The data were collected tick-by-tick from a Reuters feed over a period of 56 months from October 1992 to May 1997. The rates were de-seasonalized on an average time scale of 30 minutes according to the procedure shown in section 1.5.1 based on data from the first 12 months.



■ Figure 2.2.3 - The upper graph shows the log returns of the USDDEM rate on time intervals of 30 minutes over a period of 56 months ranging from October 1992 until May 1997 leading to 81788 data points. The two lower graphs are for comparison with a simulated normal distribution with the same variance and with a α-stable distribution with index $\alpha = 1.9$. *Source: D. Würtz (1997), unpublished.*

The logarithmic returns are shown in the upper time series of Figure 2.2.3 covering more than 80'000 data points. The simple inspection by eye tells us that an approximation of the returns

by *iid* normal random deviates with equal mean and variance or by *iid* α-stable random deviates with an appropriate index is a rather poor approach. The simple inspection of a time series by eye is quite impressive, and we see that by no means a random normal process can explain the dynamics of a financial time series. The same holds for a random α-stable process. Whereas in the first case the structure of extremal events appears to be absent, we find in the second case too many and too large peaks in the form of "needles". Also we miss the effect of the clustering of volatilities.

Figure 2.2.4: Even modelling the time series with a GARCH(1,1) model, either with normal or α-stable innovations does not yield a satisfactory result in comparison of the simulated with the empirical data.

Figure 2.2.5: That a simple GARCH(1,1) model cannot be sufficient to generate the pronounced structures in a financial time series becomes evident if we investigate the seasonal patterns in more detail. Although we have already removed the major seasonal dependencies through the de-seasonalization approach based on the business-time concept, there remain still pronounced structures in the autocorrelation function and in the periodogram as shown in the figures. Mainly the day-one peak with lag number 67 is the most evident one.

Figure 2.2.6: The periodogram suggests to try a model with 7 components with time lags as suggested by the periodogram analysis. However, the α-stable innovations overestimate the peaks in the log-returns, so that we apply innovations truncated at a predetermined peak level. Now it is quite impressive how close the simulated time series looks to the empirical foreign exchange return data.

Figure 2.2.7: The remaining question is now, how good does the modelled time series reflect the distributional properties. For this we have plotted the distribution of the log returns for the estimated models in comparison with the empirical log return data. In addition a quantile-quantile plot is shown and the distribution of monthly kurtosis values.

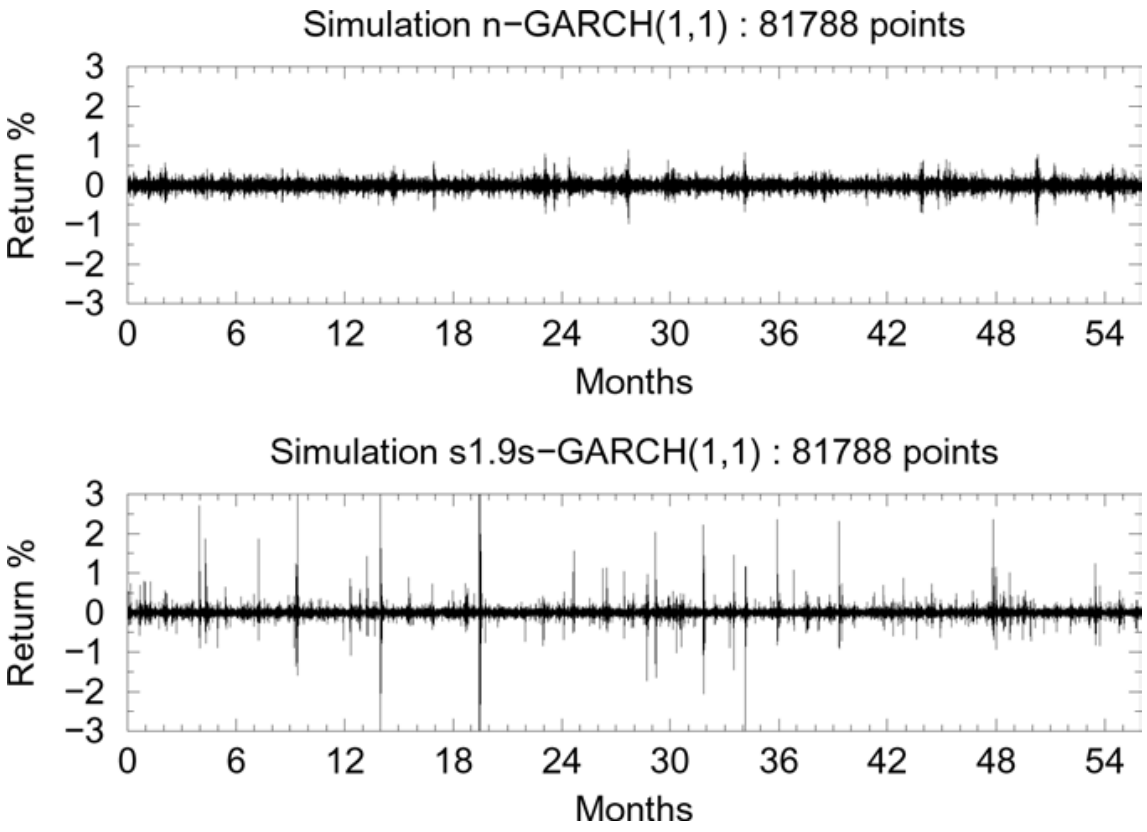
Figure 2.2.8: Now, we will take a look on the scaling properties of the log returns. The scaling power law calculated from aggregating the log return time series is best fitted by the truncated 7-component GARCH model. Considering a monthly moving time window of 1 year length we find that the scaling exponent changes significantly over time. This is true in physical time as well as in θ-time. Plotting the distribution of the scaling exponents we find the best agreement with the distribution obtained from the truncated 7-component GARCH model.

Figure 2.2.9: With the last graph we like to investigate the case of a simple GARCH(1,1) model in more detail. The graph shows the change of the parameters α, β and the persistence over time. When the scaling index approaches 0.5, the persistence becomes close to one.

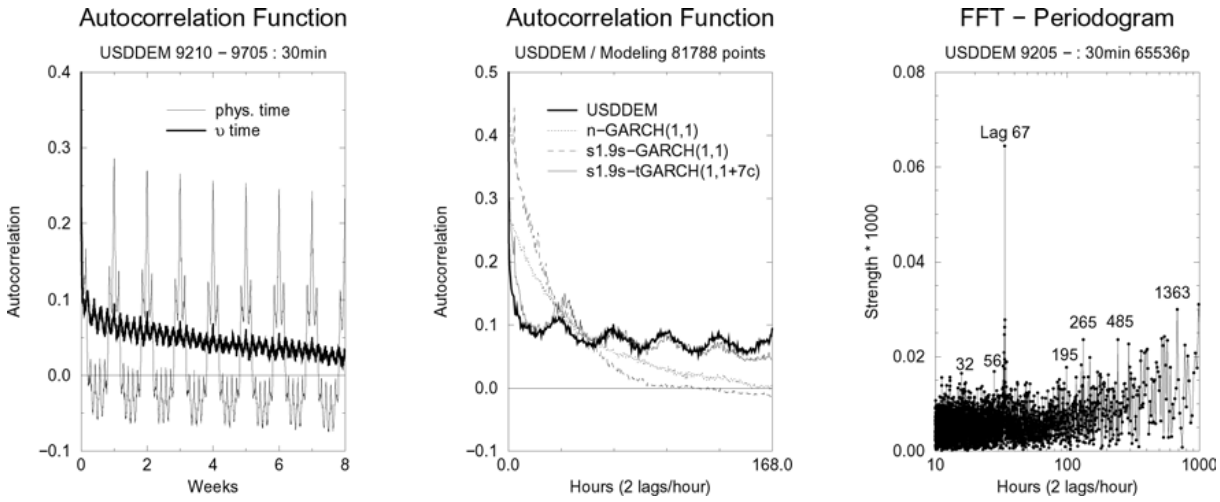
Figure 2.2.10: For one realization we show the log-likelihood function as a function of α and β in a 3-D perspective and in a 2-D contour plot. Note how flat the log-likelihood surface appears. This is a general property when estimating GARCH models, and it is a fact which requests for powerful optimization algorithms.

Exercise: USDDEM GARCH Modelling

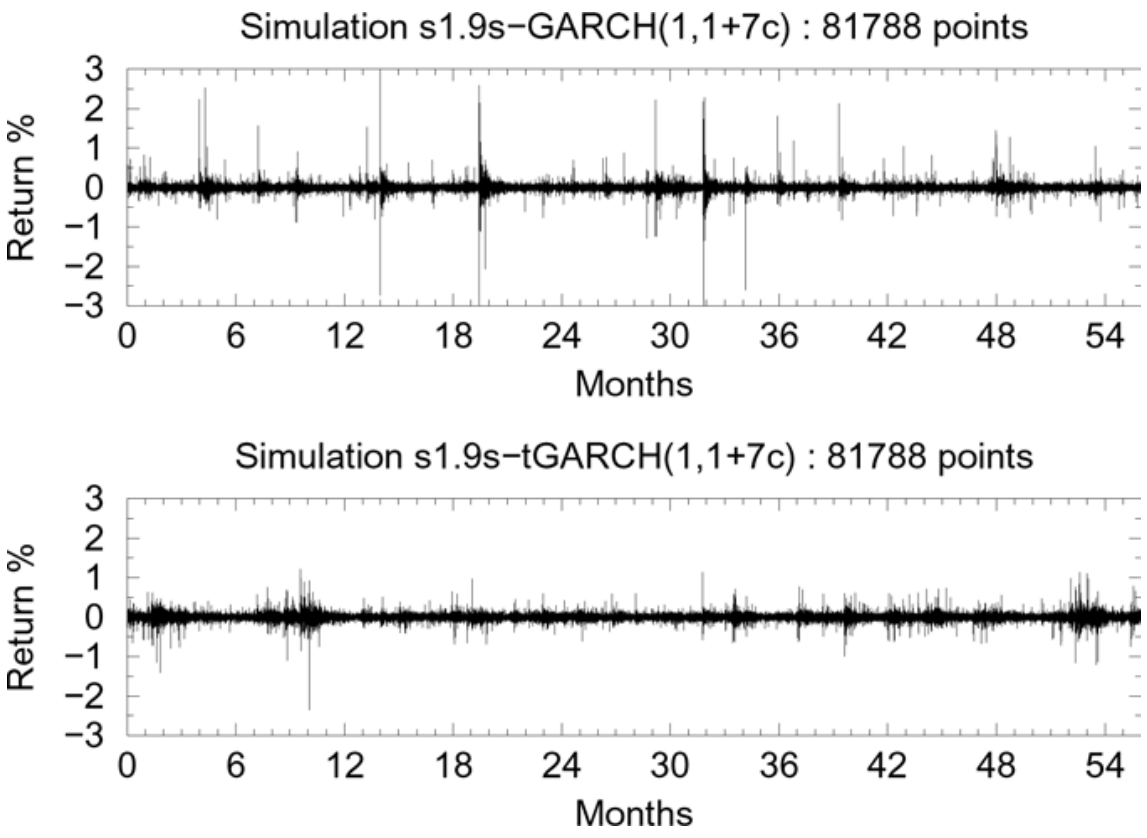
> Reinvestigate some of the aspects presented in this case study. The data records for the de-seasonalized USDDEM FX rates are part of the `fSeries` Library.



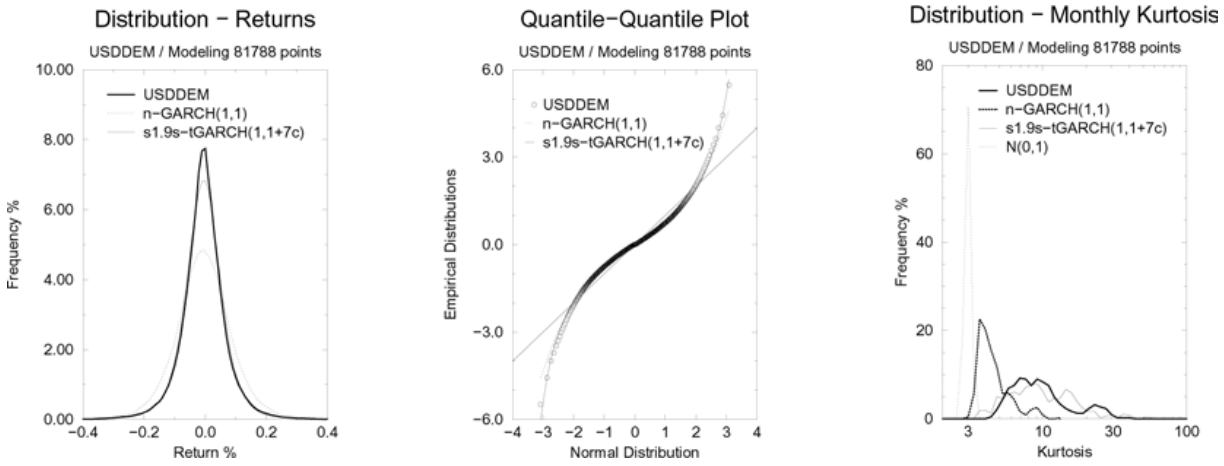
■ Figure 2.2.4 - The graphs show the simulated returns from a GARCH(1,1) model with normal distributed innovations and returns from a GARCH(1,1) model with stable distributed innovations with index $\alpha = 1.9$. *Source: D. Würtz (1997), unpublished.*



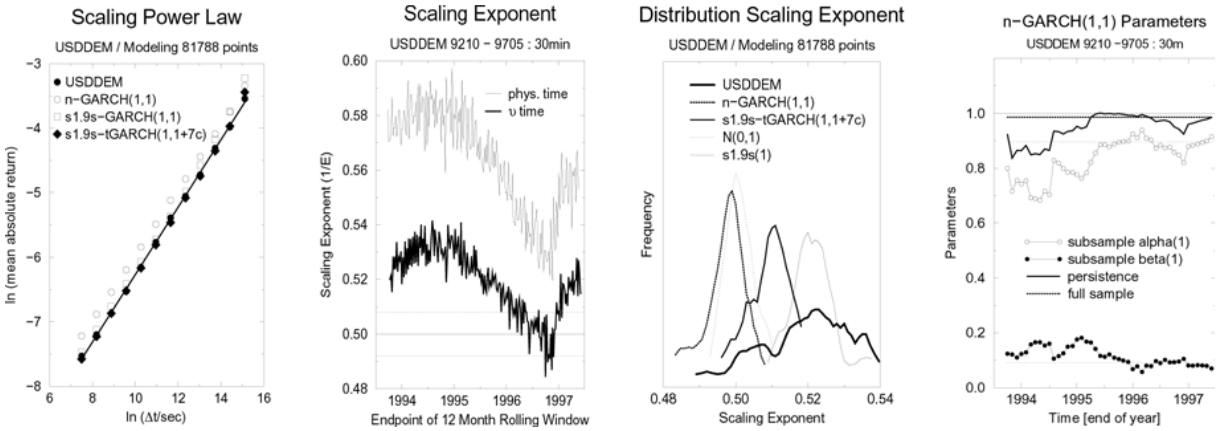
■ Figure 2.32.5 - The figure shows to the left the autocorrelation function of the USDDEM exchange rates for records sampled every 30 minutes in physical time. In the middle we have the same graph, but now the data are sampled in business time. Seven more or less pronounced structures are still remained and marked in the periodogram. *Source: D. Würtz (1997), unpublished.*



■ Figure 2.2.6 - The upper graph shows a simulated times including additionally 7 components, names 1.9-GARCH(1,1+7c) model, derived from the periodogram analysis of the USDDEM rates. The lower graph displays returns obtained from a simulation of a truncated 1.9-tGARCH(1,1+7c) model. Note, all parameters for the individual models are estimated by a log-likelihood estimation procedure. *Source: D. Würtz (1997), unpublished.*

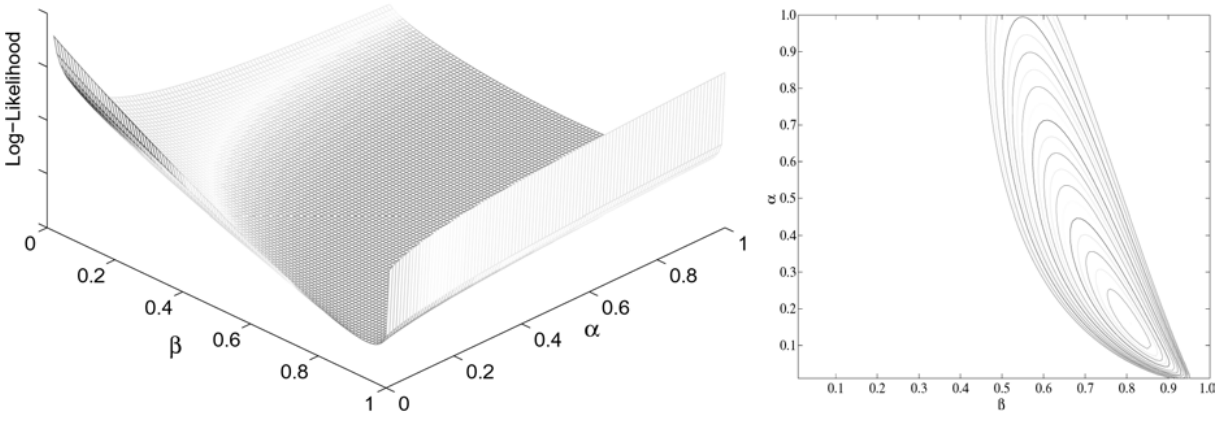


■ Figure 2.2.7 - The figure to the left shows the distribution of the empirical data in comparison to the GARCH(1,1) model with normal innovations and to the 7-component GARCH model with truncated α-stable innovations. In the middle, the QQ-Plot shows how close the truncated GARCH model fits the data. The figure to the right displays the distribution of the kurtosis as obtained from monthly moving time windows of 1 year length. The best agreement is again achieved by the truncated GARCH model. *Source: D. Würtz (1997), unpublished.*



■ Figure 2.2.8 - The first 3 graphs show the scaling law behavior of the log-returns: The scaling power law, the change of the scaling exponent over time, and the distribution of scaling exponents. The calculations were done on a monthly moving time window of 1 year length.

■ Figure 2.2.9 - The last graph to the right shows the variation of the parameters α, β and the associated persistence over time for a simple GARCH(1,1) model with normal innovations. This graph is consistent with the findings concerning the scaling law index: As the scaling index approaches 0.5, the persistence becomes close to one. *Source: D. Würtz (1997), unpublished.*



■ Figure 2.2.10 - The figure shows the log-likelihood surface for a GARCH(1,1) model with normal innovations as estimated from the USDDEM exchange rates. During the optimization ω is kept fixed through the empirical variance of the log return data. *Source: D. Würtz, unpublished.*

Notes and Comments

The original sources on ARCH and GARCH modelling are the papers *Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation* written by Engle (1982) and *Generalized Autoregressive Conditional Heteroskedasticity* written by Bollerslev (1986). An early review article *ARCH Models: Properties, Estimation and Testing* was written by Bera and Higgins (1993). Further readings include the articles collected in the book *ARCH* edited by Engle (1995). This collection brings together a series of leading research papers. Papers present both theory and financial market analysis, and discuss the key issues in the use of ARCH models to study volatility and correlation: which model to use, what time intervals to employ, how to model multivariate systems, how to apply the models to price and trade financial instruments. SThe original source for APARCH models is Ding, Granger and Engle (1993). Case studies on stocks include the work of Peters (2001) and of Lambert and Laurent (2001).

There are a series of *further aspects* we have not considered in this section. These include for example irregularly spaced ACD-GARCH models, integrated and fractionally integrated GARCH models, heterogeneous ARCH models, the temporal aggregation of ARCH and GARCH models, the testing for heteroscedasticity in time series data.

ACD-GARCH Models: One of the salient features of financial market data is that they are irregularly spaced. Thus durations between observed events of interest are themselves random and Engle and Russell (1998) proposed the autoregressive conditional duration (ACD) model to tackle this problem. Since then the empirical analysis of durations between market events has developed in several directions and moreover has integrated some aspects of the microstructure theory of financial markets. The ACD model has been extended n different directions. Jasiak (1996) analyzed the persistence of intertrade durations using the fractionally integrated ACD model, FIACD. She argues that the autocorrelation function of the durations can show a slow, hyperbolic rate of decay typical of long memory processes. Grammig and Maurer (1999) used the ACD model with a Burr distribution rather than a Weibull, thus allowing more flexibility in the shape of the conditional hazard function. Bauwens and Giot (1999) have developed a logarithmic ACD model that avoids positivity restrictions on the parameters and is therefore more flexible to introduce exogenous variables. Bauwens and Giot (1998) introduced an asymmetric ACD model where the dynamics of the duration process depend on the state of the price process. Russell and Engle (1998) also analyzed jointly durations and prices. Ghysels et al. (1997) introduced the stochastic volatility duration model (SVD). They claim that the fact the durations appear to be driven only by movements in the conditional mean is not sufficient, and they proposed a model in which the volatility of the durations is also stochastic.

IGARCH and FIGARCH Models: The GARCH model implies that the effect of a volatility shocks vanishes over time at an exponential rate. By contrast, the integrated GARCH model, IGARCH, implies a complete persistence of such a shock. These features are often in contradicition to the stylized facts drawn from the study of financial markets. To cope with this issue, Baillie, Bollerslev and Mikkelsen (1996) introduced the *Fractionally Integrated GARCH* model,

FIGARCH, that allows for some persistence of volatility shocks more in line with these facts. Empirical applications of this model to the major daily exchange rates were undertaken by Baillie, Bollerslev and Mikkelsen (1996), Tse (1998), and Beine, Laurent and Lecourt (1999).

HARCH Models: Müller et al., (1995), introduced the *Heterogeneous Auto-Regressive Conditional Heteroscedastic process*, HARCH. This process has been developed as an improvement of traditional ARCH-type models in order to describe the behavior of FX time series: (1) a long memory in the volatility with a positive autocorrelation of absolute price changes declining slower than exponentially, and (2) an asymmetry between volatilities observed with different time resolutions. The HARCH process precisely reproduces these empirical facts. The fat tail behavior of the HARCH process was investigated by Embrechts et al. (1996).

Splus/R Software: The *software* from the `fSeries` Library includes Splus functions to simulate and to estimate APARCH processes. In addition the R module `tseries` written by Trapletti also contains a `garch()` function and methods to model a basic GARCH(p,q) time series process by computing the maximum-likelihood estimates of the conditionally normal model. `garch()` uses a Quasi-Newton optimizer. The optimizer uses a Hessian approximation computed from the BFGS update. The gradient is either computed analytically or using a numerical approximation. For more details see Dennis, Gay and Welsch (1981), Dennis and Mei (1979), Dennis and More (1977), and Goldfarb (1976).

S+GARCH Module: Furthermore, a commercial S module S+GARCH is available, including functions on univariate GARCH, `garch()`, and multivariate GARCH modelling, `mgarch()`. The univariate case includes the basic GARCH model, the Threshold GARCH model, the Power GARCH model, the Exponential GARCH model, the Two-Component GARCH model, and GARCH-in-Mean models. The multivariate models are the Diagonal VECH model of Bollerslev, Engle and Woldridge (1988), the BEKK Model of Engle and Kroner (1995), the Matrix Diagonal Model of Ding (1994) and Bollerslev, Engle and Nelse (1995), the Vector Diagonal Model, the Two-Parameter Model, the Conditional Constant Correlation Model of Bollerslev (1987), and the Principal Component Model by Ding (1994) originally proposed by Kahn. In addition to providing maximum likelihood estimation for the univariate and multivariate GARCH models for a conditionally Gaussian errors distribution, S+GARCH provides conditional non-Gaussian distribution likelihood estimation based on univariate and multivariate t-distributions, and univariate generalized Gaussian distributions. The BHHH algorithm, Brent et al. (1974), is used for optimization.

2.3 Regression Modelling from the Time Series Point of View

Sir Francis Galton showed in 1886 that the height of the sons of tall fathers "regressed" towards the mean height of the population through several successive generations. In other words, sons of unusually tall fathers tend to be shorter than their fathers and sons of unusually short fathers tend to be taller than their fathers.

Introduction

Today, the term regression applies to different types of prediction problems and does not necessarily imply a regression towards the population mean. In this section we will discuss some modern regression tools from the point of view of time series analysis and forecast. The material on modern regression tools is exhaustive and thus it cannot be presented in general. Here we will concentrate on four topics: These deal with the conceptual approach of regression analysis with examples from (i) *Linear Models*, (ii) *Generalized Additive Models*, (iii) *Projection Pursuit Regression*, and (iv) *Multivariate Adaptive Regression Splines*. For all mentioned methods implementations of the algorithms are available as Splus and R functions.

We will use regression analysis mainly to find an adequate approximation of a multivariate function in general perturbed by noise. The goal is to model the dependence of a response variable y on one or more predictor variables $x_1, \ldots, x_n$, given realizations (named pattern) $\{y_i, x_{1i}, \ldots, x_{ni}\}_1^N$. The dynamics that generated the data is presumed to be described by

$$y = f(x_1, \ldots, x_n) + \epsilon \tag{2.82}$$

over some domain $(x_1, \ldots, x_n) \in \mathbb{D} \subset \mathbb{R}^n$ containing the data. $f(\cdot)$ captures the joint predictive relationship of y on $(x_1, \ldots, x_n)$ and ϵ serves as an additive stochastic component. The aim of regression analysis is to construct a functional relationship $\widehat{f}(x)$ that can serve as a *reasonable* approximation to $f(\cdot)$ over the domain $\mathbb{D}$ of interest. The notion of reasonableness depends on the purpose for which the approximation is to be used. Lack of accuracy is usually defined by the expected error

$$E = \frac{1}{N} \sum_{i=1}^{N} w(\mathbf{x}_i) \Delta[\widehat{f}(\mathbf{x}_i), f(\mathbf{x}_i)] \ . \tag{2.83}$$

Here $\mathbf{x} = (x_1, \ldots, x_n)$, Δ is some measure of distance, and $w(\mathbf{x})$ is an optional weight function. In the following we concentrate on univariate responses y, for the "multivariate" generalization we refer to the literature []. It is also worth to note, that in the case of univariate autoregressive time series analysis we will interpret equation (2.82) as

$$x_t = f(x_{t-1}, \ldots, x_{t-n}) + \epsilon_t \ . \tag{2.84}$$

The principal approach to function approximation has been to fit a parametric function $g(\mathbf{x}|\{\widehat{a}_j\}_1^p)$ to the training pattern most often by least-squares. That is

$$\widehat{f}(\mathbf{x}) = g(\mathbf{x}|\{\widehat{a}_j\}_1^p) \ , \tag{2.85}$$

where the parameter estimates are given by

$$\{\widehat{a}_j\}_1^p = \text{argmin}_{\{\widehat{a}_j\}_1^p} \sum_{i=1}^{N} [y_i - g(\mathbf{x}|\{\widehat{a}_j\}_1^p)]^2. \tag{2.86}$$

As one would expect, the *Linear Model*, LM, is the most commonly used parameterization defined through the linear function

$$g(\mathbf{x}|\{\widehat{a}_j\}_1^p) = a_0 + \sum_{i=1}^{p} a_i x_i, \quad p \leq n \ . \tag{2.87}$$

An *Additive Model*, AM, extends the notion of a linear model by allowing some or all linear functions of the predictors to be replaced by arbitrary smooth functions of the predictors. Thus the standard linear model defined by eqn. () is replaced by the additive model

$$g(\mathbf{x}|\{\widehat{a}_j\}_1^p) = a_0 + \sum_{i=1}^{p} h_i(x_i), \quad p \leq n \ . \tag{2.88}$$

Because the forms of the functions h_i are generally unknown, they are estimated using some form of a smoother. Several methods of smoothing a function are implemented in the Splus or R software packages, like for example locally weighted regression smoothers, cubic spline smoothers, or kernel type smoothers. However, the problem which arises with the AM approach is that many new model parameters will be added. Thus the dimensional setting of the models can become quite large, due to the fact that many data sets are high dimensional. Then it is a common practice to use lower dimensional linear projections of the data. Since there exist infinitely many projections from a higher to a lower dimension, it is important to have a technique of *pursuing* a finite sequence of projections that can reveal the relevant structures of the data. Combining both *projection* and *pursuit* leads to the so called *Projection Pursuit Regression*, PPR, modelling approach. The approximation is of the form

$$\widehat{f}(\mathbf{x}) = \alpha_0 + \sum_{i=1}^{p} f_i(\alpha_i^T \mathbf{x}) \ . \tag{2.89}$$

Multivariate Adaptive Regression Splines, are a technique to improve these methodologies. Its geometrical concept is to partition the whole region in disjoint subregions $\{R_m\}_1^M$ and to fit with a separate function in each subregion, generally taken to be as constants.

$$\text{if}\, \mathbf{x} \in \mathbb{R}_m \ , \quad \text{then} \quad \widehat{f}(\mathbf{x}) = g_m(\mathbf{x}|\{a_j\}_1^p) \ , \tag{2.90}$$

$$\text{with} \quad g_m(\mathbf{x}|a_m) = a_m \ . \tag{2.91}$$

2.3.1 Linear and Generalized Linear Models

The starting point in our exploration of regression models for time series analysis and forecasting will be the classical linear model approach. First we study the traditional *Linear Model* approach, discuss how to estimate the regression coefficients and how to test their significance. Modelling the dependence of a continuous response is made explicit by estimating the slope and intercept of a straight line. *Generalized Linear Models* extend the classical linear model approach to non-gaussian response probability distributions together with a link function describing how the responses are related to the linear predictor. *Logit* and *Probit Models* are discussed in some more detail.

Classical Linear Models

We start with the classical *Linear Model* approach. We collect the n responses in a column vector $\mathbf{y}$, which we view as the realization of a random vector $\mathbf{Y}$ with mean $E(\mathbf{Y}) = \mu$ and variance-covariance matrix $var(\mathbf{Y}) = \sigma^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The diagonal elements of $var(\mathbf{Y})$ are all σ^2 and the off-diagonal elements are all zero, so the n observations are uncorrelated and have the same variance. Under the assumption of normality, $\mathbf{Y}$ has a multivariate normal distribution with the standard mean and variance. Suppose we have data for the p predictors $x_1, \ldots, x_p$, which take the values $x_{i1}, \ldots, x_{ip}$ for the i-th unit. We will assume that the expected response depends on these predictors and we will assume that μ_i is a linear function of the predictors

$$\mu_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} \tag{2.92}$$

for some unknown coefficients $\beta_1, \ldots, \beta_p$. These coefficients are called *regression coefficients*.

The equation can be rewritten more compactly using matrix notation as $\boldsymbol{\mu} = \mathbf{X}\boldsymbol{\beta}$, where $\boldsymbol{\mu}$ is a column vector with all the expected responses, $\boldsymbol{\beta}$ is a column vector with all the regression coefficients, and $\mathbf{X}$ is a $n \times p$ matrix containing the values of the p predictors for the n units. The matrix $\mathbf{X}$ is usually called the *model matrix* or *design matrix*. The expression $\mathbf{X}\boldsymbol{\beta}$ is called the *linear predictor*.

How do we estimate the parameters $\boldsymbol{\beta}$ and σ^2 ? The likelihood principle instructs us to pick the values of the parameters that maximize the logarithm of the likelihood function. The normal log-likelihood becomes

$$logL(\boldsymbol{\beta}, \sigma^2) = -\frac{n}{2} log(2\pi\sigma^2) - \frac{1}{2}\sum (y_i - \mu_i)^2/\sigma^2 \ . \tag{2.93}$$

Note, that maximizing the log-likelihood with respect to the regression coefficients β for a fixed value of σ^2 is equivalent to minimizing the sum of squared differences between observed and expected values, or residual sum of squares

$$\text{RSS}(\boldsymbol{\beta}) = \sum (y_i - \mu_i)^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \ . \tag{2.94}$$

Taking the derivatives of the residual sum of squares with respect to β and setting the derivative equal to zero leads to the so called *normal equations* for the maximum log-likelihood estimator to $\widehat{\boldsymbol{\beta}}$

$$\mathbf{X^T X}\widehat{\boldsymbol{\beta}} = \mathbf{X^T y} \ . \tag{2.95}$$

If the model matrix is of full column rank (most statistical packages detect and omit redundancies automatically), we yield an explicit formula for the *ordinary least squares estimator* (OLS) or *Maximum log-Likelihood Estimator*, MLE, of the linear parameters

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X^T X})^{-1}\mathbf{X^T y} \ . \tag{2.96}$$

There are several numerical methods around for solving the normal equations, e.g. Gaussian elimination, Cholesky decomposition, model factoring, Householder reflections, Givens rotations, Gram-Schmidt orthogonalization. We will not discuss these methods in view of their advantages and disadvantages. We will trust the calculations implemented in a reliable statistical software package.

Substituting the OLS estimator of $\boldsymbol{\beta}$ into the log-likelihood function gives a profile likelihood for σ^2

$$logL(\sigma^2) = -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2}\mathrm{RSS}(\widehat{\boldsymbol{\beta}})/\sigma^2 \ . \tag{2.97}$$

Differentiating this expression with respect to σ^2 and setting the derivative to zero leads to

$$\sigma^2 = \mathrm{RSS}(\widehat{\boldsymbol{\beta}})/n \ . \tag{2.98}$$

The estimator happens to be biased, which can be corrected dividing by $n-p$ instead of n[1].

How can we test the significance of the regression coefficients β? The Wald Test allows us to test

$$H_0 : \ \beta_j = 0 \ . \tag{2.99}$$

The MLE $\widehat{\beta}_j$ has a distribution with mean 0 (under H_0) and variance given by the j-th diagonal element of $var(\widehat{\boldsymbol{\beta}}) = (\mathbf{X^T X})^{-1}\sigma^2$. Thus we can base our test on the ratio

$$t = \frac{\widehat{\beta}_j}{\sqrt{var(\widehat{\beta}_j)}} \ . \tag{2.100}$$

Under the assumption of normality of the data, the ratio of the coefficient to its normal error has under H_0 a Student's t distribution with $n-p$ degrees of freedom when σ^2 is estimated, and a standard normal distribution if σ^2 is known. In large samples the ratio has approximately a standard normal distribution.

[1]Note, that is is analogous to the use of $n-1$ instead of n when estimating a variance.

How to model the dependence of a Continuous Response y on a single linear predictor x? Let us start by recognizing that the response will vary even for constant values of the predictor, and model this fact by treating the responses y_i as realizations of random variables

$$Y_i \sim N(\mu_i, \sigma^2) \tag{2.101}$$

with means μ_i depending on the values of the predictor x_i and constant variance σ^2. The simplest way to express the dependence of the expected response μ_i, on the predictor x_i is to assume a linear functional relationship

$$\mu_i = \alpha + \beta x_i \ , \tag{2.102}$$

which defines a straight line. Equation (2.101) can be interpreted to define the random structure of the model writing equivalently $Y_i = \mu_i + \epsilon_i$, where $\epsilon_i \sim N(0, \sigma^2)$ and equation (2.102)) defines the systematic structure of the model stipulating that $\mu_i = \alpha + \beta x_i$. Combining these two statements yields the traditional formulation of the model.

It may be of interest to note that in the simple linear regression the estimates of the constant value and intercept are given by

$$\widehat{\alpha} = \overline{y} - \widehat{\beta}\overline{x} \quad \text{and} \quad \widehat{\beta} = \frac{\Sigma(\mathrm{x} - \overline{\mathrm{x}})(\mathrm{y} - \overline{\mathrm{y}})}{\Sigma(\mathrm{x} - \overline{\mathrm{x}})^2} \ , \tag{2.103}$$

where $\overline{x}$ and $\overline{y}$ are the means of the predictor and response variables, respectively. A straightforward generalization is the dependence of a continuous response on two or more linear predictors.

How to fit a linear model? The Splus function `lm(formula, data, ...)` allows to fit a linear model, and the function `predict.lm()` to forecast from new data. The function `summary.glm()` produces a summary report of a fitted `glm` object. Here are some more details:

Splus - Linear Modelling

`lm(formula, data, ...)` fits a linear model. As *arguments* for the function `lm()` serve the `formula`, a formula object, with the response on the left of a $\sim$ operator, and the terms, separated by `+` operators, on the right (e.g. `formula <- response ~ predictor.2 + predictor.5`), and `data` a data.frame in which to interpret the variables named in the formula. Several *optional arguments* can be added including for example weights, subset specifications, and others. The *returned value* is an object of class `lm` representing the fit. Generic functions such as `predict`, `summary`, `print`, `plot`, and many others have methods to show the results of the fit. The structure of the `lm` object is described in detail in the help document, and contains components like, `coefficients`, `residuals`, `fitted.values`, and many others.

`predict.lm(object, newdata, set.fit=F, ...)` extracts the fitted values from a `lm` object and returns the predictions. In the argument list `object` is a fitted `lm` object, `newdata` is a data frame containing the values at which predictions are required, and if `se.fit` is set TRUE, pointwise standard errors are computed along with the predictions.

Exercise: CAPM - Betas for Portfolio Management

The capital *asset pricing model* (CAPM) is an important model in the field of portfolio management. It explains variations in the rate of return r_j on a security j as a function of the return on a market portfolio r_m. Generally, the return on any investment is a measure relative to its opportunity cost, which is the risk-free asset return r_f . The resulting difference is called the risk premium, since it is the reward or punishment for making a risky investment. CAPM says that the risk premium on security j is proportional to the risk premium on the market portfolio.

$$r_j - r_f = \alpha_j + \beta_j(r_m - r_f) \ ,$$

where α_j represents a rate of price change and β_j is the j'th security's beta value. A stock's beta is important because it reveals the stock's volatility. It measures the sensitivity of security's return to variation in the whole stock market. As such, values of beta less than one indicate that the stock is "defensive" since its variation is less than the market's. A beta greater than one indicates an "aggressive" stock.

Download monthly prices of the 30 Dow Jones shares and the index itself from *finance.yahoo.com*, and calculate the associated log-returns. For the risk free return use the 1-Month Treasury Bills which can be downloaded from *www.economagic.com*.

First compute mean and standard deviations for each share, the index, and the short term interest rate. Plot dependent variables against the independent variable. Estimate α's and β's on a rolling window for all shares. Calculate the profit (total return), risk (volatility), and maximum drawdowns of the two portfolios where the first invests in the 10 shares with the lowest *beta*'s and the second in the 10 shares with the highest *beta*'s. How is the portfolio affected, using monthly β forecasts instead of using historical β's?

Generalized Linear Models

A *Generalized Linear Models*, GLM, extends the classical linear model and is therefore applicable to a wider range of data analysis problems. The GLM consists of the following components:

- A monotonic differentiable link function g describes how the expected value of y_i is related to the linear predictor η_i.
$$\eta_i = g(\mu_i) = \mathbf{x}_i\beta,$$
where the linear component μ_i is defined as in the case of the LM.
- The response variables y_i are independently distributed and have a probability distribution from an exponential family.
$$f(y_i) = exp\left\{\frac{y_i\theta_i - b(\theta_i)}{a_i(\phi)} + c(y_i, \phi)\right\}.$$
Here, θ_i and ϕ can be identified essentially as location and scale parameters, and $a_i(\phi)$, $b(\theta_i)$, and $c(y_i, \phi)$ are known functions. The dispersion parameter is either known or must be estimated.

The exponential family includes such useful distributions as the Normal, Binomial, Poisson, Gamma, and others. As in the case of LMs, fitted GLMs can be summarized through statistics such as parameter estimates, their standard errors, and goodness-of-fit statistics. Parameter estimates are obtained using the principle of maximum log-likelihood and hypothesis tests are based on comparisons of likelihoods.

Exercise: The Exponential Family

Show, that for the probability density function of the exponential family the mean and variance are given by

$$E(Y_i) = \mu_i = b'(\theta_i)$$
$$var(Y_i) = \sigma_i^2 = \phi b''(\theta_i) a_i(\phi)$$

Identify, for the normal distribution

$$f(y_i) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\{-\frac{1}{2}\frac{(y_i - \mu_i)^2}{\sigma^2}\}$$

the functions $a_i(\phi)$, $b(\theta_i)$, and $c(y_i, \phi)$

A GLM is constructed by choosing an appropriate response probability distribution and an appropriate link function. Here are some examples:

Classical Linear Model:
- response variable: a continuous variable
- distribution: Normal
- link function: identity $\eta \equiv \mu$

Poisson Regression in Log Linear Model:
- response variable: a count
- distribution: Poisson
- link function: logarithm $\eta = log(\mu)$

Gamma Model with Log Link:
- response variable: a positive continuous variable
- distribution: Gamma
- link function: logarithm $\eta = log(\mu)$

Logistic Regression:
- response variable: a proportion
- distribution: Binomial
- link function: logit $\eta = log(\frac{\mu}{1-\mu})$

How to estimate the parameters? - The IRLS Algorithm: An important practical feature of GLMs is that they can all be fit to data using the same algorithm: *Iteratively Reweighted Least Squares*, IRLS. Given a trial estimate of the parameters $\widehat{\boldsymbol{\beta}}$ we calculate the estimated linear predictor $\widehat{\eta}_i = \mathbf{x_i^T}\widehat{\boldsymbol{\beta}}$ and use that to obtain the fitted values $\widehat{\mu}_i = g^{-1}(\widehat{\eta}_i)$. Using these quantities, we calculate the working dependent variable

$$z_i = \widehat{\eta}_i + (y_i - \widehat{\mu}_i)\frac{d\eta_i}{d\mu_i} \ , \tag{2.104}$$

where the rightmost term is the derivative of the link function evaluated at the trial estimate. Next we calculate the iterative weights

$$w_i = p_i / \left[b''(\theta_i) \left(\frac{d\eta_i}{d\mu_i} \right)^2 \right] \ , \tag{2.105}$$

where $b''(\theta_i)$ is the second derivative of $b(\theta_i)$ evaluated at the trial estimate and we have assumed that $a_i(\phi)$ has the usual form ϕ/p_i. This weight is inversely proportional to the variance of the

working dependent variable z_i given the current estimates of the parameters, with proportionality factor ϕ. Finally, we obtain an improved estimate of $\boldsymbol{\beta}$ regressing the working dependent variable z_i on the predictors $\mathbf{x_i}$ using the weights w_i, i.e. we calculate the weighted least-squares estimate

$$\boldsymbol{\beta} = (\mathbf{X^T W X})^{-1} \mathbf{X^T W z} \ , \tag{2.106}$$

where $\mathbf{X}$ is the model matrix, $\mathbf{W}$ is a diagonal matrix of weights with the entries w_i and $\mathbf{z}$ is the response vector with entries z_i. The procedure is repeated until convergence is achieved.

Example: Estimation with Normal Data

Assume normal data with identity link $\eta_i = \mu_i$. Then the derivative is $d\eta_i/d\mu_i = 1$ and the working dependent variable is y_i itself. Since in addition $b''(\theta_i) = 1$ and $p_i = 1$, the weights are constant and no iteration is required.

What is a Quantal-Response Model? - Logit, Probit and Extreme Value Distribution: In a quantal-response model, a strictly increasing and continuous distribution function F is given such that F has range $(0, 1)$. The responses y_i have value 0 or 1. The probability that $y_i = 1$ is $\mu_i = F(\eta_i)$. The link function $g(\cdot)$ is F^{-1}. Common examples involve the *Logit Model* based on the logistic distribution function

$$F(x) = \frac{1}{1 + e^{-x}} \ ,$$

the *Probit Model* based on the standard normal distribution function,

$$F(x) = \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy \ ,$$

and the extreme-value model based on the distribution function

$$F(x) = 1 - exp(-e^x) \ .$$

In the logit case, the link function is the logit transformation with value $log[p/(1-p)]$ for p in $(0, 1)$. In the probit case, the link function is Φ^{-1}. In the extreme-value case, the link function has value $log[-log(1-p)]$ for p in $(0, 1)$.

How to fit a Generalized Linear Model? The Splus function `glm(formula, family, data, ...)` allows to fit a generalized linear model, and the function `predict.glm()` to forecast from new data. The function `summary.glm()` produces a summary report of a fitted `glm` object. Here are some more details:

Splus - Generalized Linear Modelling

`glm(formula, family, data, ...)` fits a generalized linear model. As *arguments* for the function `glm()` serve as in the case of the linear model the `formula` expression, the data.frame `data`, and in addition the `family` object, a list of functions and expressions for defining the link and variance functions, initialization and iterative weights. Families supported are `gaussian`, `binomial`, `poisson`, `Gamma`, `inverse.gaussian` and `quasi`. Functions like binomial produce a family object, but can be given without the parentheses. Family functions can take arguments, as in `binomial(link=probit)`

in which to interpret the variables named in the formula. Several additional *optional arguments* can be added including for example weights, subset specifications, and others. The *returned value* is an object of class `glm` which inherits from the `lm` object representing the fit. Generic functions such as `predict`, `summary`, `print`, `plot`, and many others have methods to show the results of the fit. The structure of the `glm` object is described in detail in the help document, and contains components like, `coefficients`, `residuals`, `fitted.values`, and many others.

`predict.glm(object, newdata, set.fit=F, ...)` extracts the fitted values from a `glm` object and returns the predictions. In the argument list `object` is a fitted `glm` object, `newdata` is a data frame containing the values at which predictions are required, and if `se.fit` is set TRUE, pointwise standard errors are computed along with the predictions.

Exercise: GLM Forecasting US Recession - `xmpRegRecession`

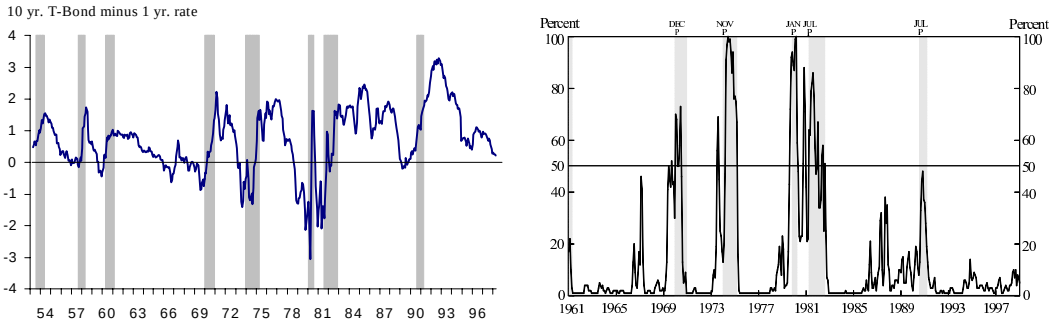
Recession is associated with bad times like high unemployment and low production, or more general with a stagnant economy. Strictly speaking though, a recession is the period when overall economic activity is actually declining and production, employment, and sales are falling below normal. A recession starts just after a business cycle peak, the high point in the level of economic activity, and ends at the business cycle trough, the low point.

A popular rule of thumb is that two consecutive quarterly declines in real GDP signal a recession. This rule is consistent with the dispersion and duration requirements for a recession and with the average recessionary path of real GDP; however, two very small quarterly declines might not produce the depth required for a recession. The most widely accepted determination of business cycle peaks and troughs is made by the *National Bureau of Economic Research*, NBER. NBERs definition of a recession is: "... a significant decline in activity spread across the economy, lasting more than a few months, visible in industrial production, employment, real income, and wholesale and retail trade", refers to the length of the recession, which must be a sustained decline.

NBER had identified six recessions from January 1960 through September 1999: 1969.12-1970.11, 1973.11-1975.03, 1980.01-1980.07, 1981.07-1982.11, and 1990.07-1991.03. Several modelling approaches are used to analyze and forecasting these periods: Simple rules of thumb using the Index of Leading Indicators, CLI, Neftci's model improving these rules by developing a a formal statistical model of the probability of recession, the Probit model improving on Neftci's model by allowing to assess the importance of multiple indicators simultaneously, the GDP forecasting model, and the Stock-Watson model that tries to capture the institutional process of the NBER's Business Cycle Dating Commitee, for details see A.J. Filardo (1999).

Here we consider the observation that the yield curve can serve as a predictor of US Recessions, e.g. see A. Estrella and F.S. Mishkin [1996]. The steepness of the yield curve should be a reasonable indicator of a possible future recession for several reasons: Current monetary policy has a significant influence on the yield curve spread and hence on real activity over the next several quarters. A rise in the short rate tends to flatten the yield curve as well as to slow real growth in the near term. This relationship, however, is only one part of the explanation for the yield curves usefulness as a forecasting tool. Expectations of future inflation and real interest rates contained in the yield curve spread also seem to play an important role in the prediction of economic activity. The yield curve spread variable examined here corresponds to a forward interest rate applicable from three months to ten years into the future. This rate can be decomposed into expected real interest rate and expected inflation components, each of which may be helpful in forecasting. The expected real rate may be associated with expectations of future monetary policy and hence of future real growth. Moreover, because inflation tends to be positively related to activity, the expected inflation component may also be informative about future growth.

Use the Probit model to directly relate the probability of being in a recession to the yield curve spread between the 10-year treasury bond and the 3-month treasury bill. Analyze the model and evaluate one, three and six month ahead forecasts. Compare the results with the Stock Watson Recession Index, shown in the figure.

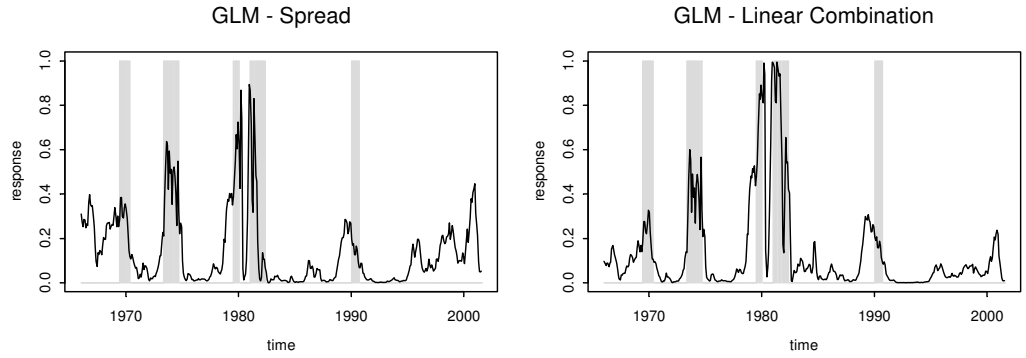


◀ Figure 2.3.1: A negative yield curve is a strong recession signal. The graph shows the difference between the yields in the 10-year and 1-year treasury bonds, which has turned negative prior to every recession since the early 1950s. However, the index turned negative in the mid-60s, but was not followed by a recession. *Source A. König, FED, (1999).*

▶ Figure 2.3.2: The figure shows the Stock Watson Index from 1961 to 1998 (black line) together with the recession periods (grey bars). *Source NBER.*

Here comes the Splus implementation:

```
# Read the Dataset from File:
# Column 1: Date - CCYYMM
# Column 2: Recession 0 | 1
# Column 3: 3 Month Treasury Bills
# Column 4: 10 Years Treasury Bonds
recession <- recession.dat()

# Settings:
horizon <- 6
n <- length(recession[,1])

# Response and Predictors
response <- recession[,2][(1+horizon):n]
predictor1 <- recession[,3][1:(n-horizon)]
predictor2 <- recession[,4][1:(n-horizon)]
predictor3 <- predictor1 - predictor2

# Mid-Month Dates:
ccyy <- floor(recession.dat[,1]/100)
time <- ccyy + (recession[,1]-100*ccyy-0.5)/12
time <- time[1:(n-horizon)]

# Data:
data <- data.frame(cbind(response, predictor1, predictor2, predictor3))

# Fit:
family <- binomial(link=probit)

# GLM Model 1 - Spread:
model.glm <- glm(response ~ predictor3, family=family, data=data)
summary.glm(model.glm)
in.sample <- predict.glm(model.glm, newdata=data, type="response")
plot(time, response, type="n", main="GLM - Spread")
lines(time, response, type="h", col=10)
lines(time, in.sample)
```



◄ Figure 2.3.3: Model 1 shows the fit of recession data based on the spread of 10 years and 1 year treasury bonds.
► Figure 2.3.4: Model 2 is more general and fits recession data, by regression of 10 years and 1 years treasury bonds instead of using the spread.

```
# GLM Model 2 - Linear Combination:
model.glm <- glm(response ~ predictor1 + predictor2, family=family, data=data)
summary.glm(model.glm)
in.sample <- predict.glm(model.glm, newdata=data, type="response")
plot(time, response, type="n", main="GLM")
lines(time, response, type="h", col=10)
lines(time, in.sample)
```

Here is the output of the Splus example program:

```
Call: glm(formula = response ~ predictor3, family=family, data=data)
Deviance Residuals:
           Min      1Q  Median      3Q   Max
         -2.12 -0.5442 -0.2827 -0.1266 3.059
Coefficients:
                    Value Std. Error t value
      (Intercept) -0.4807    0.09989  -4.813
       predictor3  0.6528    0.08229   7.933
(Dispersion Parameter for Binomial family taken to be 1 )
      Null Deviance: 353.8 on 426 degrees of freedom
Residual Deviance: 261.9 on 425 degrees of freedom
Number of Fisher Scoring Iterations: 5
Correlation of Coefficients:
            (Intercept)
     predictor3 0.4956


Call: glm(formula = response ~ predictor1 + predictor2, family=family, data=data)
Deviance Residuals:
           Min      1Q  Median      3Q   Max
        -3.197 -0.3981 -0.2558 -0.1058 2.184
Coefficients:
                    Value Std. Error t value
      (Intercept) -2.3922    0.32148  -7.441
       predictor1  0.6923    0.08871   7.804
       predictor2 -0.4534    0.08536  -5.312
(Dispersion Parameter for Binomial family taken to be 1 )
      Null Deviance: 353.8 on 426 degrees of freedom
Residual Deviance: 217 on 424 degrees of freedom
Number of Fisher Scoring Iterations: 5
Correlation of Coefficients:
             (Intercept) predictor1
     predictor1 -0.0918
     predictor2 -0.3349     -0.8986
```

2.3.2 AM - Additive and Generalized Additive Models:

The *Additive Model*, AM, generalizes the LM by

$$\hat{f}(\mathbf{x}) = s_0 + \sum_{i=1}^{p} s_i(x_i) \ , \tag{2.107}$$

where the s_j are unknown smooth functions. These functions are not given in a parametric form, but instead they have to be estimated from the data avoiding the assumption of linearity in the explanatory variables. However, AM retains the assumption that explanatory variable effects are additive. Thus the response is modelled as the sum of arbitrary smooth univariate functions of the explanatory variables. Note, that one needs a reasonably large sample size to estimate each s_i.

A *Generalized Additive Model*, GAM, extends the AM in the same spirit as the GLM extends the LM model, namely for allowing a link function and for allowing non-normal distributions from the exponential family.

GAMs and GLMs can be applied in similar situations, but they serve different analytic purposes. GLMs emphasize estimation and inference for the parameters of the model, while GAMs focus on exploring data non-parametrically. In this sense GAMs are more suitable for exploring the data set and visualizing the relationship between the dependent variable and the independent variables.

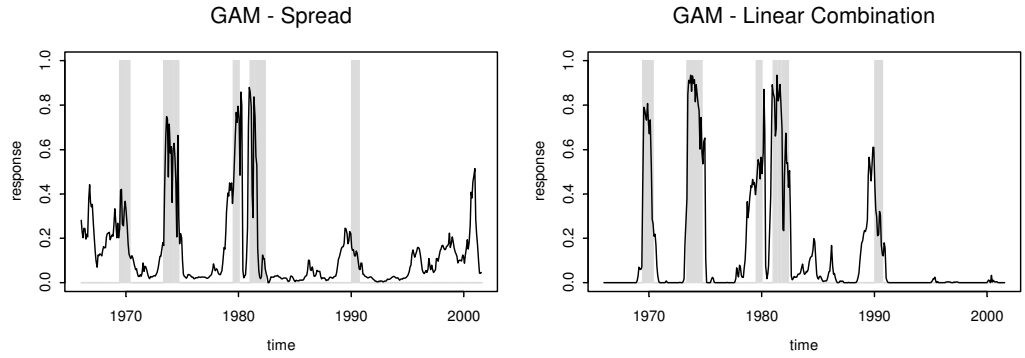
How to estimate the parameters? - Backfitting and Scoring: A GAM model can be fitted using the local scoring algorithm, which iteratively fits weighted additive models by backfitting. The backfitting algorithm is a Gauss-Seidel method for fitting additive models, by iteratively smoothing partial residuals. The algorithm separates the parametric from the nonparametric part of the fit, and fits the parametric part using weighted linear least squares within the backfitting algorithm. The algorithm requires a smoothing operation as for example a kernel or a spline smoother. For large classes of smoothing functions, the backfitting algorithm converges to a unique solution.

How to fit a Generalized Additive Model? The Splus function `gam(formula, family=gaussian, data, ...)` allows to fit a generalized additive model. The function `summary.gam()` produces a summary report of a fitted `gam` object, the function `predict.gam()` forecasts from new data, and the function `plot.gam()` creates an appropriate plot for each term in a generalized additive model object. Here are some more details:

Splus - Generalized Additive Modelling

`gam(formula, family, data, ...)` fits a generalized additive model. As *arguments* for the function `gam()` serve as in the case of the generalized linear model the `formula` expression, the `family` name, and the data.frame `data`. Several *optional arguments* can be added including for example weights, subset specifications, and others. The *returned value* is an object of class `gam` which inherits from the `glm` and `lm` object representing the fit. Generic functions such as `predict`, `summary`, `print`, `plot`, and many others have methods to show the results of the fit. The structure of the `gam` object is described in detail in the help document, and contains components like, `coefficients`, `residuals`, `fitted.values`, and many others.

`predict.gam(object, newdata, set.fit=F, ...)` provides a "safe" method of prediction from a fitted `gam` object. In the argument list `object` is a fitted `gam` object, `newdata` is a data frame



◄ Figure 2.3.5: Model 1 shows the fit of recession data based on the spread of 10 years and 1 year treasury bonds.
► Figure 2.3.6: Model 2 is more general and fits recession data, by regression of 10 years and 1 years treasury bonds instead of using the spread.

containing the values at which predictions are required, and if `se.fit` is set TRUE, pointwise standard errors are computed along with the predictions.

Exercise: GAM Forecasting US Recession, cont. - `xmpRegRecession`

Here we continue with the US Recession example and show how to fit and forecast the data within the generalized additive modelling approach. For smoothing we apply local regression, `lo`, alternatively we can select spline smoothing, `s`.

```
# Settings ...
family <- binomial(link=probit)

# GAM Model 1 - Spread:
model.gam <- gam(response ~ lo(predictor3), family=family, data=data)
summary.gam(model.gam)
in.sample <- predict.gam(model.gam, newdata=data, type="response")
plot(time, response, type="n", main="GAM - Spread")
lines(time, response, type="h", col=10)
lines(time, in.sample)

# GAM Model 2 - Linear Combination:
model.gam <- gam(response ~ lo(predictor1) + lo(predictor2), family=family, data=data)
summary.gam(model.gam)
in.sample <- predict.gam(model.gam, newdata=data, type="response")
plot(time, response, type="n", main="GAM")
lines(time, response, type="h", col=10)
lines(time, in.sample)
```

The fit is shown in figure 2.3.5 and figure 2.3.6. Here is the output of the Splus example program, produced by the function `summary.gam()`.

```
Call: gam(formula=response~lo(predictor3), family=family, data=data)
Deviance Residuals:
       Min      1Q  Median      3Q   Max
    -2.061 -0.5187 -0.2697 -0.1855 2.811
(Dispersion Parameter for Binomial family taken to be 1 )
    Null Deviance: 353.8 on 426 degrees of freedom
Residual Deviance: 252.8 on 422.1 degrees of freedom
```

```
Number of Local Scoring Iterations: 5
DF for Terms and Chi-squares for Nonparametric Effects
                  Df Npar Df Npar Chisq  P(Chi)
      (Intercept)  1
   lo(predictor3)  1     2.9       9.37 0.02273


Call: gam(formula=response~lo(predictor1)+lo(predictor2), family=family, data=data)
Deviance Residuals:
       Min      1Q   Median        3Q   Max
   -2.111 -0.2208 -0.02788 -0.000622 2.034
(Dispersion Parameter for Binomial family taken to be 1 )
    Null Deviance: 353.8 on 426 degrees of freedom
Residual Deviance: 134.9 on 418 degrees of freedom
Number of Local Scoring Iterations: 10
DF for Terms and Chi-squares for Nonparametric Effects
                  Df Npar Df Npar Chisq    P(Chi)
      (Intercept)  1
   lo(predictor1)  1     3.5      32.54 7.78e-007
   lo(predictor2)  1     2.5      38.78 9.00e-009
```

2.3.3 Projection Pursuit Regression

The AM considers sums of functions taking arguments in the natural coordinates of the space of explanatory variables. When the underlying function is additive with respect to variables formed by linear combinations of the original explanatory variables, the AM is inappropriate. Projection Pursuit Regression can handle such cases.

Projection Pursuit Regression, PPR, uses an approximation of the form

$$\hat{f}(\mathbf{x}) = \alpha_0 + \sum_{j=1}^{p} f_j(\boldsymbol{\alpha}_j^T \mathbf{x}) \ . \tag{2.108}$$

Thus the model is approached as a sum of nonlinear transformations of one-dimensional projections of $\mathbf{x}$, where in principle any smoother may be used to estimate the functions f_j. In practice, the PPR model is most useful in situations where p can be chosen as smaller than d, in which case a "dimension reduction" has taken place. The effectiveness of the approach lies in the fact that even for small to moderate p, many classes of functions can be closely fit by approximations of this form.

A *Generalized Projection Pursuit Regression Model*, GPPR, extends the PPR to the exponential family distributions and a link function in the same spirit as the GLM and GAM extend the Linear Model and the Additive Model. This approach is discussed in detail by O. Lingjaerde and K. Liestol (1998). In the following we will concentrate on the PPR algorithm.

How to estimate the Parameters? We will present a way for finding the f_j, in a nonparametric regression context. Friedman and Stuetzle (1981) present the following algorithm.

Algorithm: Projection Pursuit Regression

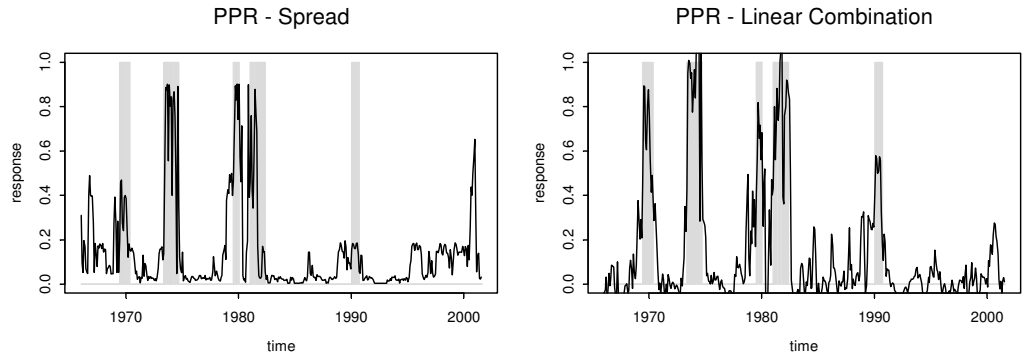
- Let $r_i = y_i$ for $i = 1, \ldots, n$ and $M = 0$. M is a "term counter".
- Construct the next term in the model. Let $\mathbf{Z} = \alpha_j^T \mathbf{x}$, and construct a smooth function $f_j(Z)$ of the current residuals ordered in ascending value of Z. Next, calculate a measure of fit $I(\alpha)$ based on the fraction of so far unexplained variance that is explained by f_j:

  $$I(\alpha) = 1 - \frac{\Sigma_{i=1}^n (r_i - f_j(\boldsymbol{\alpha}_j^T \mathbf{x}_i))^2}{\Sigma_{i=1}^n r_i^2}$$

  Find the vector a_{M+1} that maximizes $I(\alpha)$, which is the heart of the projection pursuit concept: $\alpha_{M+1} = argmax_\alpha I(\alpha)$ and the smooth function f_{M+1} .
- If $I(\alpha)$ becomes smaller than a specified threshold, the algorithm is converged. Otherwise, we define new residuals and proceed to the next step: $M = M + 1$ and $r_i = r_i - f_{M+1}(\boldsymbol{\alpha}_{M+1} \mathbf{x}_i)$, for $i = 1, \ldots, n$.

Note that the functions f_j are univariate. Essentially, PPR breaks down the data into dimensions, not necessarily along the standard coordinate axes, but axes determined by $I(\alpha)$. Moreover, by virtue of the algorithm's termination criterion, as few dimensions as possible are used to describe the data.

How to fit a PPR model? The PPR algorithm is implemented under Splus by the function `ppreg(y, x, ...)`. Unfortunately, the arguments do not allow for the quiet common `formula` input. `ppr(formula, data...)` is the counterpart under R, distributed in the `modreg` package. `ppr` is based on essentially the same code as used by `ppreg` under Splus. The `fSeries` library makes use of the `modreg` package under Splus. Here are some more details:



◄ Figure 2.3.7: Model 1 shows the fit of recession data based on the spread of 10 years and 1 year treasury bonds.
► Figure 2.3.8: Model 2 is more general and fits recession data, by regression of 10 years and 1 years treasury bonds instead of using the spread.

Splus - Projection Pursuit Regression

`ppr(formula, family, data, ...)` fits a projection pursuit regression model. As *arguments* for the function `ppr()` serve as in the case of GLM and GAM the `formula` expression, and the data.frame `data`. Several *optional arguments* can be added including for example weights, subset specifications, and others. The *returned value* is an object of class `ppr` representing the fit. Generic functions such as `predict`, `summary`, `plot`, and many others have methods to show the results of the fit. The structure of the `ppr` object is described in detail in the help document, and contains components like, `residuals`, and many others.

`predict.ppr(object, newdata, set.fit=F, ...)` extracts the values from a `ppr` object and returns the predictions. In the argument list `object` is a fitted `ppr` object, `newdata` is a data frame containing the values at which predictions are required.

Exercise: PPR Forecasting US Recession, cont. - `xmpRegRecession`

Here we continue with the US Recession example and show how to fit and forecast the data within the projection pursuit regression modelling approach.

```
# Settings ...

# PPR Model 1 - Spread:
model.ppr <- ppr(response ~ predictor3, data=data)
summary.ppr(model.ppr)
in.sample <- predict.ppr(model.ppr, newdata=data, type="response")
plot(time, response, type="n", main="PPR - Spread")
lines(time, response, type="h", col=10)
lines(time, in.sample)

# PPR Model 2 - Linear Combination:
model.ppr <- ppr(response ~ predictor1 + predictor2, data=data)
summary.ppr(model.ppr)
in.sample <- predict.ppr(model.ppr, newdata=data, type="response")
plot(time, response, type="n", main="PPR")
lines(time, response, type="h", col=10)
lines(time, in.sample)
```

The fit is shown in figure 2.3.7 and figure 2.3.8. Here is the output of the Splus example program, produced by the function `summary.ppr()`.

```
Call: ppr.formula(formula = response ~ predictor3, data=data, nterms=5)
Goodness of fit:
    5 terms
    0
Projection direction vectors:
    term 1 term 2 term 3 term 4 term 5
         1     -1      1      0      0
Coefficients of ridge terms:
    term 1   term 2   term 3   term 4   term 5
  0.206790 0.024804 0.007813 0.000000 0.000000


Call: ppr.formula(formula = response ~ predictor1 + predictor2, data=data, nterms=5)
Goodness of fit:
    5 terms
      15.63
Projection direction vectors:
                term 1  term 2  term 3  term 4  term 5
    predictor1  0.8652  0.4321 -0.5235 -0.7428  0.6379
    predictor2 -0.5014 -0.9018 -0.8520  0.6695 -0.7701
Coefficients of ridge terms:
     term 1 term 2 term 3 term 4 term 5
     0.2405 0.1810 0.1973 0.1141 0.1509
```

2.3.4 MARS - Multivariate Adaptive Regression Splines

This section describes the *Multivariate Adaptive Regression Spline*, MARS, approach developed by J.H. Friedman (1990). The goal of this procedure is to improve the existing methodologies outlined above. The material is divided into three parts. First we present the geometrical concept of recursive partition regression and the corresponding stepwise regression approach. This connection allows us easily to understand the MARS algorithm outlined in the second part. The third and final part will be devoted to software software implementation of the MARS algorithm together with some examples and exercises.

Recursive Partitioning Regression

The Geometrical Concept: MARS is most easily understood through its connections with *Recursive Partitioning Regression*. This model takes the form

$$\text{if}\,\mathbf{x} \in \mathbb{R}_m, \quad \text{then} \quad \widehat{f}(\mathbf{x}) = g_m(\mathbf{x}|\{a_j\}_1^p) \ , \tag{2.109}$$

where $\{R_m\}_1^M$ are disjoint subregions representing a partition of $\mathbb{D}$. The functions g_m are generally taken to be as constants

$$g_m(\mathbf{x}|a_m) = a_m \ . \tag{2.110}$$

The goal is to use the data to simultaneously estimate a good set of subregions and the parameters associated with the separate functions in each subregion. Continuity at subregion boundaries is not enforced.

How to partition the domain? The partitioning is accomplished through the recursive splitting of previous subregions. The starting region is the entire domain $\mathbb{D}$. At each stage of the partitioning all existing subregions are optimally split in two daughter subregions. The recursive subdivision is continued until a large number of subregions are generated. The subregions are then recombined in a reverse manner until an optimal set is reached. This process must be based on criterion that penalizes both for lack of fit and increasing number of regions. Thus recursive partitioning models are fairly interpretable, can be represented by a binary tree, and are fairly rapid to construct and especially rapid to compute. However, the disadvantage of the approximating functions is that they are discontinuous at the subregion boundaries, and this is in many cases more than a cosmetic problem.

Stepwise Regression: Recursive partitioning regression is generally viewed as a geometrical procedure. This framework provides the best intuitive insight into its properties. However, it can also be viewed in a more conventional light as a stepwise regression procedure. The idea is to produce an equivalent model to equations (2.109) and (2.110) by replacing the geometrical concepts of regions and splitting with the arithmetic notions of adding and multiplying. The starting point is to cast the approximation (2.109) and (2.110) in the form of an expansion in a set of basis functions

$$\widehat{f}(\mathbf{x}) = \sum_{m=1}^{M} B_m(\mathbf{x}) \ . \tag{2.111}$$

The basis functions B_m take the form

$$B_m(\mathbf{x}) = I\left[x \in \mathbb{R}_m\right] , \tag{2.112}$$

where I is an indicator function having the value one if its argument is true and zero otherwise. The $\{a_m\}_1^M$ are the coefficients of the expansion whose values are jointly adjusted to give the best fit to the data. The $\{R_m\}_1^M$ are the same subregions of the predictors space as in equations (2.109) and (2.110). Since these regions are disjoint only one basis function is nonzero for any point $\mathbf{x}$ so that (2.111) and (2.112) are equivalent to (2.109) and (2.110).

The aim of recursive partitioning is not only to adjust the coefficient values to best fit the data, but also to derive a good set of basis functions (subregions) based on the data at hand. A forward stepwise regression procedure is outlined as:

Algorithm 1 - Recursive Partitioning

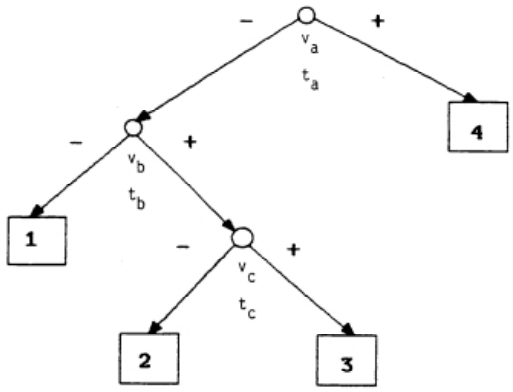
01 $B_1(\mathbf{x}) \leftarrow 1$
02 For $M = 2$ to M_{max} do: $lof^\star \leftarrow \infty$
03 For $m = 1$ to $M - 1$ do:
04 For $v = 1$ to n do:
05 For $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$
06 $g \leftarrow \Sigma_{i \neq m} a_i B_i(\mathbf{x}) + a_m B_m(\mathbf{x}) H[+(x_v - t)] + a_M B_m(\mathbf{x}) H[-(x_v - t)]$
07 $lof \leftarrow \min_{a_1,\ldots,a_M} LOF(g)$
08 if $lof < lof^\star$ then $lof^\star \leftarrow lof$; $m^\star \leftarrow m$; $v^\star \leftarrow v$; $t^\star \leftarrow t$; end if
09 end for
10 end for
11 end for
12 $B_M(\mathbf{x} \leftarrow B_{m^\star} H[-(x_v - t)]$
13 $B_{m^\star}(\mathbf{x} \leftarrow B_{m^\star} H[+(x_v - t)]$
14 end for
15 end algorithm 1

■ Algorithm 1: $H[\eta]$ denotes a step function indicating a positive argument being 1 for $\eta \geq 0$ and 0 otherwise, and $lof(g)$ is some procedure that computes the lack-of-fit of a function $g(\mathbf{x})$ to the data. The first line is equivalent to setting the initial region to the entire domain. The first For-loop iterates the "splitting" procedure with M_{max} being the final number of regions (basis functions). The next three (nested) loops perform an optimization to select a basis function $B_{m^\star}$ (already in the model), a predictor variable $x_{v^\star}$ and a "split point" $t^\star$. The quantity being minimized is the lack-of-fit of a model with $B_{m^\star}$ being replaced by its product with the step function $H[+(x_{v^\star} - t^\star)]$, and with the addition of a new basis function which is the product of $B_{m^\star}$ and the reflected step function $H[-(x_{v^\star} - t^\star)]$. This is equivalent to splitting the corresponding region $R_{m^\star}$ on variable $v^\star$ at split point $t^\star$. Note that the minimization of $LOF(g)$ with respect to the expansion coefficients (line 7) is a linear regression of the response on the current basis function set.

The basis functions produced by Algorithm 1 have the form

$$B_m(\mathbf{x}) = \prod_{k=1}^{K_m} H[s_{km} \cdot (x_{v(k,m)} - t_{km})] \ . \tag{2.113}$$

The quantity K_m is the number of "splits" that gave rise to B_m whereas the arguments of the step functions contain the parameters associated with each of these splits. The quantities s_{km} in (2.113) take on values ± 1, and indicate the (right/left) sense of the associated step function.



$$B_1 = H[-(x_{v_a} - t_a)]H[-(x_{v_b} - t_b)]$$
$$B_2 = H[-(x_{v_a} - t_a)]H[+(x_{v_b} - t_b)]H[-(x_{v_c} - t_c)]$$
$$B_3 = H[-(x_{v_a} - t_a)]H[+(x_{v_b} - t_b)]H[+(x_{v_c} - t_c)]$$
$$B_4 = H[+(x_{v_a} - t_a)]$$

■ Figure 2.3.9: Owing to the forward stepwise (recursive) nature of the procedure the parameters for all the basis functions can be represented on a binary tree that reflects the partitioning history. The figure 2.3.1 shows a possible result of running Algorithm 1 in this binary tree representation, along with the corresponding basis functions. The internal nodes of the binary tree represent the step functions and the terminal nodes represent the final basis functions. Below each internal node are listed the variable v and location t associated with the step function represented by that node. The sense of the step function s is indicated by descending either left or right from the node. Each basis function (2.113) is the product of the step functions encountered in a traversal of the tree starting at the root and ending at its corresponding terminal node.

The $v(k, m)$ label the predictor variables and the t_{km} represent values on the corresponding variables.

With most forward stepwise regression procedures it makes sense to follow them by a backwards stepwise procedure to remove basis functions that no longer contribute sufficiently to the accuracy of the fit. This is especially true in the case of recursive partitioning. In fact the strategy here is to deliberately overfit the data with an excessively large model, and then to trim it back to proper size with a backwards stepwise strategy. A proper tree pruning scheme must delete (sibling) regions in adjacent pairs by merging them into a single (parent) region. Thus splits rather than regions (basis functions) are removed.

The MARS Algorithm

Continuity: A fundamental limitation of recursive partitioning models is lack of continuity. The models produced by equations (2.109) and (2.110) are piecewise constant and sharply discontinuous at subregion boundaries. This lack of continuity severely limits the accuracy of the approximation. It is possible, however, to make a minor modification to Algorithm 1 which will cause it to produce continuous models with continuous derivatives.

The only aspect of Algorithm 1 that introduces discontinuity into the model is the use of the step function $H[\eta]$ as its central ingredient. If the step function were replaced by a continuous

function of the same argument everywhere it appears (lines 6, 12, and 13), algorithm 1 would produce continuous models. The choice for a continuous function to replace the step function is guided by the fact that the step function as used in Algorithm 1 is a special case of the one-sided truncated power basis functions for representing q-th order splines

$$b_q(x-t) = (x-t)_+^q \ . \tag{2.114}$$

Here t is the knot location, q is the order of the spline, and the subscript indicates the positive part of the argument. For $q > 0$ the spline approximation is continuous and has $q-1$ continuous derivatives. A two-sided truncated power basis is a mixture of functions of the form

$$b_q^{\pm}(x-t) = [\pm(x-t)]_+^q \ . \tag{2.115}$$

The step functions appearing in Algorithm 1 are seen to be two-sided truncated power basis functions for $q = 0$ splines.

The usual method for generalizing spline fitting to higher dimensions is to employ basis functions that are tensor products of univariate spline functions. Using the two-sided truncated power basis for the univariate functions, these multivariate spline basis functions take the form

$$B_m^{(q)}(\mathbf{x}) = \prod_{k=1}^{K_m} [s_{km} \cdot (x_{v(k,m)} - t_{km})]_+^q \ , \tag{2.116}$$

along with products involving the truncated power functions with polynomials of lower order than q. Note that $s_{km} = \pm 1$. Comparing equation (2.113) with (2.116) we see that the basis functions (2.113) produced by recursive partitioning are a subset of a complete tensor product ($q = 0$) spline basis with knots at every (distinct) marginal data point value. Thus:

> *Recursive partitioning can be viewed as a forward/backward stepwise regression procedure for selecting a (relatively very small) subset of regressor functions from this (very large) complete basis.*

MARS Implementation: Algorithm 2 implements the forward stepwise part of the MARS strategy by incorporating the following modifications to recursive partitioning:

> **Modifications to Algorithm 1**
>
> (a) Replacing the step function $H[\pm((x-t)]$ by a truncated power spline function $[\pm(x-t)]_+^q$.
> (b) Not removing the parent basis function $B_{m^\star}(x)$ after it is split, thereby making it and both its daughters eligible for further splitting.
> (c) Restricting the product associated with each basis function to factors involving distinct predictor variables.

The proposed implementing strategy is to employ $q = 1$ splines in the analog of Algorithm 1 in lines 6, 12, and 13. This procedure, called "forward stepwise algorithm", is outlined as:

Algorithm 2 - Forward Stepwise

01 $B_1(\mathbf{x}) \leftarrow 1\,;\; M \leftarrow 2$
02 Loop until $M > M_{max}$: $lof^\star \leftarrow \infty$
03 For $m = 1$ to $M - 1$ do:
04 For $v! \in \{v(k,m) | 1 \leq k \leq K_m\}$
05 For $t \in \{x_{vj} | B_m(\mathbf{x}_j > 0\}$
06 $g \leftarrow \Sigma_{i=m}^{M-1} a_i B_i(\mathbf{x}) + a_M B_m(\mathbf{x}) H[+(x_v - t)]_+ + a_{M+1} B_m(\mathbf{x}) H[-(x_v - t)]_+$
07 $lof \leftarrow \min_{a_1,\ldots,a_{M+1}} LOF(g)$
08 if $lof < lof^\star$ then $lof^\star \leftarrow lof$; $m^\star \leftarrow m$; $v^\star \leftarrow v$; $t^\star \leftarrow t$; end if
09 end for
10 tend for
11 end for
12 $B_M(\mathbf{x} \leftarrow B_{m^\star} H[+(x_v - t)]_+$
13 $B_{M+1}(\mathbf{x} \leftarrow B_{m^\star} H[-(x_v - t)]_+$
14 end loop
15 end algorithm 2

■ Algorithm 2: The parent basis function is included in the modified model in line 6, and remains in the updated model through the logic of lines 12-14. Basis function products are constrained to contain factors involving distinct variables by the control loop over the variables in line 4, see 2.113, 2.116. This algorithm produces M_{max} $q = 1$ tensor product (truncated power) spline basis functions that are a subset of the complete tensor product basis with knots located at all distinct marginal data values. As with recursive partitioning, this basis set is then subjected to a backwards stepwise deletion strategy to produce a final set of basis functions. The knot locations associated with this approximation are then used to derive a piecewise cubic basis, with continuous first derivatives, thereby producing the final (continuous derivative) model.

Unlike recursive partitioning, the basis functions produced by Algorithm 2 do not have zero pairwise product expectations; that is, the corresponding "regions" are not disjoint but overlap. Removing a basis function does not produce a "hole" in the predictor space (so long as the constant basis function B_1 is never removed). As a consequence, it is not necessary to employ a special "two at a time" backward stepwise deletion strategy based on sibling pairs. A usual "one at a time" backward stepwise procedure of the kind ordinarily employed with regression subset selection can be used. Algorithm 3 presents such a procedure for use in the MARS context.

Algorithm 3 - Backwards Stepwise

01 $J^\star = \{1, 2, \cdots, M_{max}\}\,;\; K^\star \leftarrow J^\star$
02 $lof^\star \leftarrow \min_{\{a_j | j \in J^\star\}} LOF(\Sigma_{j \in J^\star} a_j B_j(\mathbf{x}))$
03 For $M = M_{max}$ to 2 do: $b \leftarrow \infty$ $L \leftarrow K^\star$
04 For $m = 2$ to M do: $K \leftarrow L - \{m\}$
05 $lof \leftarrow \min_{\{a_k | k \in K\}} LOF(\Sigma_{k \in K} a_k B_k(\mathbf{x}))$
06 if $lof < b$ then $b \leftarrow lof$ $K^\star \leftarrow K$; end if
07 if $lof < lof^\star$ then $lof^\star \leftarrow lof$; $J^\star \leftarrow K$; end if
08 end for
09 end for
15 end algorithm 3

■ Algorithm 3: Initially (line 1) the model is comprised of the entire basis function set $J^\star$ derived from Algorithm 2. Each iteration of the outer For-loop of the algorithm causes one basis function to be deleted. The inner For-loop chooses which one. It is the one whose removal either improves the fit the most or degrades it the least. Note that the constant basis function $B_1(x) = 1$ is never eligible for removal. The algorithm constructs a sequence of $M_{max} - 1$ models, each one having one less basis function than the previous one in the sequence. The best model in this sequence is returned (in $J^\star$) upon termination.

ANOVA *Decomposition:* The result of applying Algorithms 2 and 3 is a model of the form

$$\widehat{f}(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m \prod_{k=1}^{K_m} [s_{km} \cdot (x_{v(k,m)} - t_{km})]_+. \tag{2.117}$$

Here a_0 is the coefficient of the constant basis function B_1, and the sum is over the basis functions B_m, equation (2.116), produced by Algorithm 2 that survive the backwards deletion strategy of Algorithm 3 and $s_{km} = \pm 1$. This (constructive) representation of the model does not provide very much insight into the nature of the approximation. By simply rearranging the terms, however, one can cast the model into a form that reveals considerable information about the predictive relationship between the response y and the predictors x. The idea is to collect together all basis functions that involve identical predictor variable sets.

The MARS model, equation (2.117), can be recast into the form

$$\widehat{f}(\mathbf{x}) = a_0 + \sum_{K_m=1} f_i(x_i) + \sum_{K_m=2} f_{ij}(x_i, x_j) + \sum_{K_m=3} f_{ijk}(x_i, x_j, x_k) + \ldots. \tag{2.118}$$

The first sum is over all basis functions that involve only a single variable. The second sum is over all basis functions that involve exactly two variables, representing (if present) two-variable interactions. Similarly, the third sum represents (if present) the contributions from three-variable interactions, and so on. Thus interpretation of the MARS model is greatly facilitated through its ANOVA decomposition (2.118). This representation identifies the particular variables that enter into the model, whether they enter purely additively or are involved in interactions with other variables, the level of the interactions, and the other variables that participate in them.

Model Selection: Several aspects of the MARS procedure (Algorithms 2 and 3) have yet to be addressed. Among these are the lack-of-fit criterion lof, and the maximum number of basis functions M_{max}. The lack-of-fit criterion used with the algorithm depends on the distance (loss) function Δ specified within the expected error functional, (2.83). The most often specified distance is squared-error loss

$$\Delta[\widehat{f}(\mathbf{x}), f(\mathbf{x})] = [\widehat{f}(\mathbf{x}) - f(\mathbf{x})]^2 \tag{2.119}$$

because its minimization leads to algorithms with attractive computational properties. We use a generalized cross-validation criterion (GCV)

$$lof(\widehat{f}_M) = GCV(M) = \frac{1}{N} \sum_{i=1}^{N} [y_i - \widehat{f}_M(\mathbf{x}_i)]^2 / [1 - \frac{C(M)}{N}]^2. \tag{2.120}$$

Here the dependencies of $f(\cdot)$, equation (2.117), and the criterion, on the number of (non-constant) basis functions M is explicitly indicated. The GCV criterion is the average-squared residual of the fit to the data (numerator) times a penalty (inverse denominator) to account for the increased variance associated with increasing model complexity $C(M)$ (number of basis functions M).

Software Implementations

The original Fortran MARS 3.5 software package written by Friedman which implements the described MARS algorithms is no longer available for a free download from the Internet. MARS is now offered as a commercial software product by Salford Systems. However, there are two other Splus/R software packages available which implement the MARS algorithm.

The MDA *Package:*

This package provides the Splus functions `mars()` and `predict.mars()` which were coded by Trevor Hastie and Robert Tibshirani from scratch, and did not use any of Friedman's MARS code. The authors claim that the functions give quite similar results to Friedman's program, but not exactly the same results. Friedman's ANOVA decomposition is not implemented nor categorical predictors are handled properly yet.

The POLYMARS *Package:*

Charles Kooperberg and Martin O'Connor (1997) have written Splus/R functions which implement the MARS procedure. In their model the regression data is generated from the "true model" given by: $y = f(X_1, \ldots, X_p) + \epsilon$. The function $f(\cdot)$ is approximated using functions that depend on only one or two of the X_i. That is, we use the approximative model (2.118), motivated from the perspective of the ANOVA decompositions in which higher order interactions are ignored. POLYMARS uses linear splines and their tensor products to model the functions $f(\cdot)$.

Initially the constant function a_0 in equation (2.118) is fitted to the data. Then the algorithm decides which basis functions are candidates for addition and which basis functions can be added to the current model. The best candidates are added, then the model is fitted and evaluated, and if the model is better than the best one, it will be saved. This procedure continues until the maximum model size is reached or no candidates are left. In the second step it is decided which basis function can be removed from the model, and the one that is the worst predictor is removed. The model will be fitted and evaluated again, and if the model is better than the best one, it will be saved. This procedure continues until the minimum model size is reached. The GCV criterion (2.120) is used, where $C(M) = d \times M$ is used. d, penalizing for larger models, is usually set a value ranging between 3 and 5. This criterion is evaluated at the end of each addition and deletion step and the best-model-so-far by this criterion is stored.

Arguments to POLYMARS*:* The `polymars()` functions has as only required arguments the responses and predictors, but offers a series of optional arguments. Here follows a brief summary, for the details we refer to the help page and the software manual.

- `responses` - vector of responses, or a matrix for multiple response regression
- `predictors` - matrix of predictor variables for the regression
- `weights` - vector of observation weights
- `maxsize` - maximum number of basis functions allowed to grow to in the stepwise addition procedure
- `gcv` - parameter used to find the overall best model from a sequence of fitted models
- `additive` - should the fitted model be additive in the predictors?
- `startmodel` - the first model that is to be fit by the polymars function
- `knots` - defines how the function is to find potential knots for the spline basis functions
- `knot.space` - is an integer describing the minimum number of order statistics apart that two knots can be
- `ts.resp` - testset responses for model selection
- `ts.pred` - testset predictors
- `ts.weights` - testset observation weights

- `classify` - when the response is discrete (categorical), polymars can be used for classification
- `factors` - used to indicate that certain variables in the predictor set are categorical variables
- `tolerance` - measure for each possible candidate to be added/deleted
- `verbose` - when set TRUE, the function will print out a line for each addition or deletion stage

Interpreting the returned model - The visible model: Using `summary` or `print` (which defaults to `summary`) on an object returned from `polymars` prints out three components:

- The **call** which produced the object.
- An Splus data-frame, **fitting** which contains certain statistics about the model fitting routine. Each row represents one step in the fitting routine. The first column *0/1* has a 1 for an addition step and a zero for a deletion step. The second column, *size*, contains the resulting number of basis functions in the model after this step. Next there is a column *RSS* containing the residual sum of squares. For multiple response regression or classification there will be more than one column *RSS1*, *RSS2* ..., one for each response or class. The last column contains the measure of fit which used to find the best overall model. It is normally *GCV* for generalized cross validation, see Section. But it can also be *T.S. RSS* for test set residual sum of squares or *T.S.M.C.* for test set misclassification.
- The POLYMARS `model` itself is printed as a data-frame, each row corresponding to a basis function. The first row corresponds to the intercept. The first four (or five) columns relate to the basis function and the last column (more than one for multiple response regression or classification) contains the coefficients. For classification coefficients see also *conversion* below.
  The first column *pred1* contains the index of the first predictor of the basis function. Column *knot1* is a possible knot in this predictor. If this column is NA, the first component is linear. If any of the basis functions of the model are categorical then there will be a *level1* column. Column *pred2* is the possible second predictor involved (if it is NA the basis function only depends on one predictor). Column *knot2* contains the possible knot for the predictor *pred2*, and it is NA when this component is linear. For example the following model

```
  pred1 knot1 level1 pred2 knot2      coefs      SE
1     0    NA     NA     0    NA  59.123913  9.0277
2    13    NA     NA     0    NA  -5.508833  2.2838
3    13  6.29     NA     0    NA   4.834013  1.9678
4     4    NA      1     0    NA   9.486980  2.6050
```

  has an intercept of 59.12 and predictor 13 has two terms in the model, a linear term with coefficient -5.51 and a term with a knot $(X13 - 6.29)_+$ with coefficient 4.83. One level of variable 4 is in the model (this was actually a 0/1 variable) with coefficient 9.49. Standard errors for the coefficients are also included.
  A line such as

```
pred1 knot1 pred2 knot2      coefs      SE
    2     0     3  0.28   12.45667  3.3382
```

  corresponds to a basis function $X2 \times (X3 - 0.28)_+$ with coefficient 12.46.

Interpreting the returned model - The invisible model:

- `model.size` contains the number of basis functions in the returned model.
- `fitted` and `residuals` contain the fitted values and the residuals of the original dataset used to fit the model.
- `responses` contains number of responses per case in original dataset.
- `ranges.and.medians` is a $3 \times p$ matrix, where p is the number of predictors in the original dataset, each column corresponding to a predictor. Rows 1 and 2 contain the minimum and maximum values of the predictor and row 3 contains the median value of the predictor. These are computed from the original (training) dataset and are used in the plotting function.

- `factor.matrix` is is a $r \times s$ matrix where r is the number of categorical predictor variables in the model and s is the maximum number of levels that any of these has $+2$. Each column represents a categorical variable. The first row contains the index of the predictor, the second row contains the number of levels it has and the remaining rows contain the value of each level (filled out to the end with NA's if necessary). This is used in the plotting function.
- `conversion` is a $c \times 2$ matrix where c is the number of classes or categories in the response when POLYMARS is used as a classifier. In classification a single response vector is split up into a multiple response matrix with each column corresponding to a class and each case having a 1 in the column corresponding to it's original response and all other columns zero.
  In the `conversion` matrix each row corresponds to one class. In the first column it's original class, as a character or number, is recorded. The second column holds the response number, or column index of the new response matrix. The coefficients of **model** are ordered according to this numbering. It is used for computing further fitted values (classes).

Several examples are discussed in Friedman's paper. Here we pick up one which was also discussed in Kooperberg's and O'Connor's (1997) POLYMARS software manual.

Example: A function of ten Variables - `xmpPolymars10`

This example shows the ability of the MARS procedure to find structure in data while ignoring noise. The data is created using the function

$$f(\mathbf{x}) = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon$$

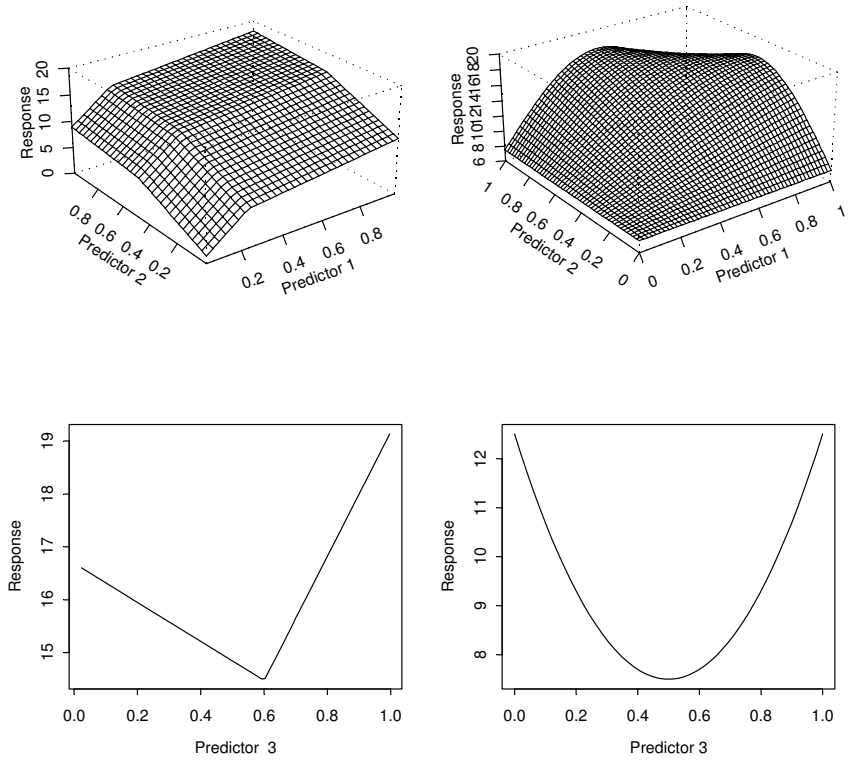
in a 10-dimensional hypercube using 100 points. The predictors where drawn from a uniform distribution and ϵ is from a standard normal distribution.

```
# Function to Estimate:
    fx <- function(x){10*sin(pi*x[,1]*x[,2])+20*(x[,3]-0.5)^2+10*x[,4]+5*x[,5]}
# Estimate Polymars Model:
    N <- 100
    x <- matrix(runif(N*10), byrow=T, ncol=10)
    y <- func1(x) + rnorm(N)
    model <- polymars(responses=y, predictors=x)
# Print Polymars Summary:
    summary.polymars(model)
# Plot Polymars z(x1,x2) and z(x3)
    plot.polymars(model, 1, 2, zlim=c(6,20))
    x <- y <- seq(0, 1, length=50)
    z <- function(x,y) {10*sin(pi*x*y)+15/2}
    persp(x=x, y=y, outer(x, y, FUN=z), zlim=c(6,20),
        xlab="Predictor 1", ylab="Predictor 2", zlab="Response")
    plot.polymars(model1, 3)
    plot(x=x, y=20*(x-0.5)^2+15/2, type="l", xlab="Predictor 3", ylab="Response")
```

The result is as follows. Note that the procedure works quite well in picking up the main structure present in the data without picking up any of the 5 noise covariates.

```
Call:
polymars(responses = y, predictors = x)

Model fitting

   0/1 size    RSS     GCV
 1   1    1 1996.2  21.660
 2   1    2 1471.9  17.390
..   .    .    ...     ...
48   0    2 1471.9  17.390
49   0    1 1996.2  21.660
```

```
Model produced

   pred1  knot1 pred2  knot2   coefs     SE
 1     0     NA     0     NA   2.106 1.0106
 2     2     NA     0     NA  -4.383 2.9119
 3     4     NA     0     NA   8.261 0.6285
 4     1     NA     0     NA  -1.414 3.9449
 5     5     NA     0     NA   5.446 0.5892
 6     2 0.4618     0     NA   9.700 4.9703
 7     1     NA     2     NA  52.752 7.4072
 8     1     NA     2 0.4618 -36.462 8.1528
 9     1 0.3165     0     NA   3.964 5.4778
10     1 0.3165     2     NA -40.366 8.6617

Rsquared : 0.879
```

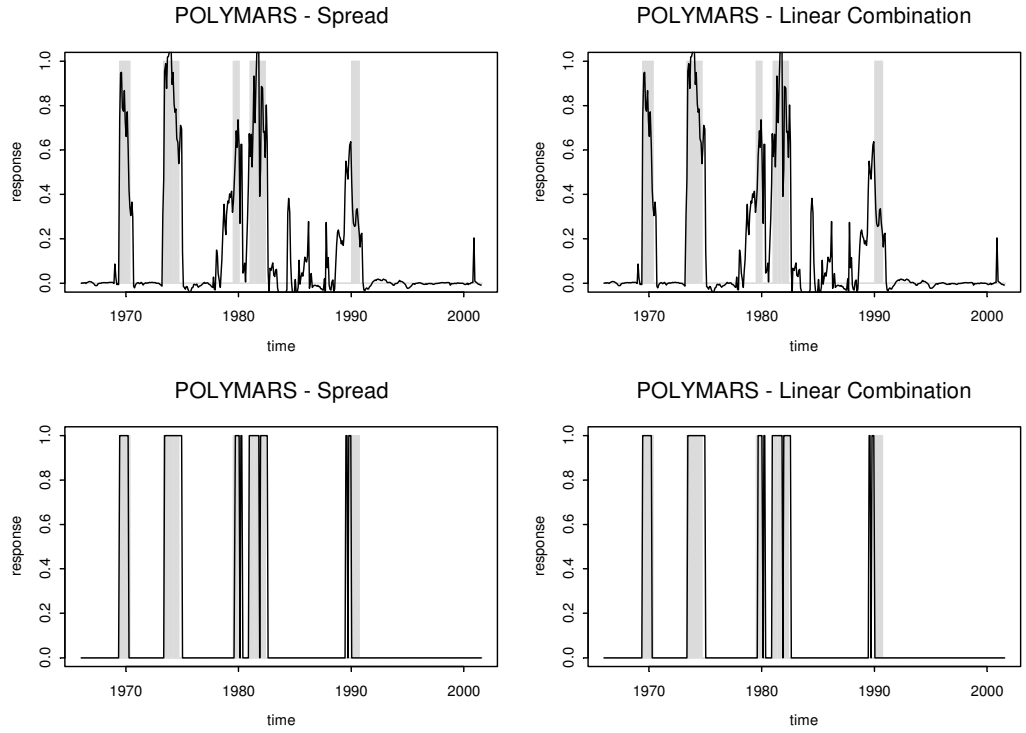


■ Figure 2.3.10: The graphs compare the POLYMARS estimate (left) to the exact functional relationship (right), using the Splus functions `plot.polymars()`, `persp()` and `plot()`.

Kooperberg and O'Connor (1997) note, that this example with its highly curved relationships is not an ideal setting for POLYMARS with it's piecewise linear splines. This is particularly true for x_3 since the procedure insists that a linear basis function is the first term fit to any variable. A linear term will not improve the fit any more than just the constant intercept term so the model building procedure will be averse to adding any x_3 terms. For this reason x_3 should be specified to be in the `startmodel`. Thus recalculate the example with the following statement

```
model <- polymars(responses=y, predictors=x, startmodel=c(3,NA,0,0))
```

and compare the result using the defaults for the `startmodel`.



◄ Figure 2.3.11: Model 1 shows the fit of recession data based on the spread of 10 years and 1 year treasury bonds. The lower graphs show the result for the classifyer POLYMARS.
► Figure 2.3.12: Model 2 is more general and fits recession data, by regression of 10 years and 1 years treasury bonds instead of using the spread. The lower graphs show the result for the classifyer POLYMARS.

Exercise: PPR Forecasting US Recession, cont. - `xmpRegRecession`

Here we continue with the US Recession example and show how to fit and forecast the data within the POLYMARS modelling approach[2]. Note, there is also the possibility to run POLYMARS in a classifyer mode, setting the argument `classify=T`. The fit is shown in figure 2.3.11 and figure 2.3.12, and for the classifyer type in figure 2.3.13 and 2.3.14.

```
# POLYMARS Model 1 - Spread:
model.polymars <- polymars(responses=response, predictors=predictors3, classify=F)
summary.polymars(model.polymars)
in.sample <- predict.polymars(model.polymars, newdata=data, type="response")
plot(time, response, type="n", main="POLYMARS - Spread")
lines(time, response, type="h", col=10)
lines(time, in.sample)

# POLYMARS Model 2 - Linear Combination:
predictors12 <- cbind(predictor1, predictor2)
model.polymars <- polymars(responses=response, predictors=predictors12, classify=F)
summary.polymars(model.polymars)
in.sample <- predict.polymars(model.polymars, newdata=data, type="response")
plot(time, response, type="n", main="POLYMARS - Linear Combination")
lines(time, response, type="h", col=10)
lines(time, in.sample)
```

[2]Note, that a formula input for the response and predictors is not available.

2.3.5 Case Study: Technical Analysis of Stock Markets

Studies on trading financial market instruments, called among traders "Technical Analysis" are in most cases based on daily data records providing Open, High, Low, and Close Prices, and sometimes also Volume and Open Interest from the Futures Markets. Such investigations can be found in most of the many trading books usually written by "Market Gurus". The simple approach presented in these books takes the signals from trading indicators to buy and sell instruments. Other investigations, inspired by "academic traders" use more elaborate approaches, where neural networks are the most popular ones. In these investigations the trading indicators are used as predictor variables for the input units of a connectionist network, and the output unit represents a value supporting the buy or sell decision.

As a typical publication in this field we cite the paper *Using AI in Developing Market Predictions: A Study of the Pacific Basin Capital Markets* written by C.C. Yang, S.T. Chou, W.H.Chu, F.Lai (1995). The authors focus on the stock markets in Hong Kong, Taiwan, and Japan in the Years 1992 and 1993 and use a neural network model to derive trading decision rules. Predictors are formed from retrospective features of the stock markets and the predictive trend is used as response. Before we further specify their investigation we make an excursion to learn more about technical trading indicators.

Trading Indicators

Technical Analysis is based on information obtained from trading indicators derived from market data records. The purpose of charting this information as function of time is to identify "price trends" and nontrending periods as they begin to develop and to make trading decisions based on this kind of information.

VOL - Trading Volume

> Volume is defined as the number of units traded during a time period. This number is significant in that it supports a prevailing price trend. Volume should expand in the direction of a major trend. If the trend is up, the volume should increase as buying pressure exceeds selling pressure. As prices are accepted and level off, there will also be a levelling in volume. The same would be true if prices fall. Volume will expand, representing a major change in the trend.

ROC - The Momentum or Rate of Change

> Momentum measures the price's "rate of change". The momentum is measured by calculating price differences over a predefined time period. As these prices oscillate around zero, they measure the rates of ascent or descent of an instrument. If prices are rising and the momentum line is above the zero line and rising, the current up-trend is accelerating. In an upward trend, if the momentum line begins to flatten out, current advances are the same as the advances achieved n number of periods (days) ago. If the momentum line begins to drop toward the zero line, this would indicate that the upward trend is still in existence but is losing momentum. When the momentum line moves below the zero line, the latest close is below the close from the number of periods one has specified. A downward trend is now in effect. Momentum can be used as a leading indicator to reveal the reversal of a trend. The momentum line shows this advance or decline and levels off while the trend is still in effect. Then it begins to move in the opposite direction as prices begin to level off. Many traders use the zero line to indicate buy and sell signals. When the momentum indicator is above the zero line a buy signal is indicated and when it's below the line is a sell signal is indicated. However, buy positions should only be taken when the momentum goes above zero if the overall market trend is up. Sell positions should be taken only when momentum goes below zero if the overall price trend is down.

EMA - Exponential Moving Averages

The moving average is a way of calculating the average price of an instrument over a given time period. As prices change over time, the average price reflects the change, but at a slower pace. The moving average is used as an indicator to identify changes in trends. The moving average compares the current price to the moving average price. If the current price moves above the moving average, expectations are higher than the average of the last n number of periods (days). This would indicate a buy signal. If the current price moves below the moving average, the current expectations are lower than the last n number of days. This would indicate a sell signal. The moving average was designed to inform the trader of an instrument's price trend, not to determine the top or bottom . Since the moving average indicates a trend, a trader would be buying or selling shortly after the current price goes through the moving average. Finding the right number of periods on which to base the moving average is critical in determining it's predictability. This time period should fit the trading cycle the trader follows. For example, a long-term trader using day charts would use a different time period than a short-term trader who uses minute or tick charts.

RSI - Relative Strength Index

The relative strength index is used by the traders to demonstrate the inner strength of a price trend. This indicator is also used with futures spread trading to track the spread between nearby and distant contract months. Whether or not a trader does spread trading, by tracking the RSI one can often see which way the market is going, as well as its strength or weakness. Relative Strength is calculated by averaging up and down closes over a given period of time. The result indicates how much strength is left in a trend. It is plotted on a scale of 0 to 100% (or equivalently between 0 and 1, sometimes the indicator is shifted to get around zero). When the RSI tops above 70%, it indicates the instrument is "overbought". When it bottoms below 30%, it indicates the instrument is "oversold". These RSI tops and bottoms will usually be formed before the underlying price indicates the same. The number of periods (days) used to measure this index will be determined after some experimentation. This number was originally recommended as 14 periods. Some analysts also use 9 or 25. Usually, the faster the instrument moves, the smaller the numbers.

MACD - Moving Average Convergence/Divergence

The MACD displays the relationship between two moving averages of prices. A 26-period exponential moving average is subtracted from a 12-period exponential moving average. Then a 9-period moving average of the MACD is plotted on top of the MACD. This is considered the "signal line", which predicts the crossing (convergence) of the two moving averages. The results are plotted around a zero line. When the MACD crosses above its signal line, a buy signal is indicated. When the MACD crosses below the signal line, a sell signal is indicated. The 26, 12 and 9 periods represent a long-term trading instrument. There are two other ways to use the MACD: – (1) When the MACD crosses above zero, the trend is up. When the MACD crosses below zero, the trend is down. (2)) When the MACD is making new lows while prices fail to reach new lows or when the MACD is making new highs and the prices fail to reach new highs a divergence occurs. – MACD is a good study for verifying what prices are doing during a long trend. During a shorter trend, the MACD may not be as reliable.

%K and %D - The Stochastic Oscillator

The Stochastic Oscillator compares a security's closing price to its price range over a specific period of time. The Stochastic Oscillator is displayed as two lines. The main line called %K and the second line %D which is an exponentially smoothed %K. The %D line renders the buy/sell signals. The result is measured as a percentage and plotted on a scale between 0 and 100%. A result above 70%, a high value of %K, would put the price near the top of the total price range. A result below 30%, a low %K, would put the price near the bottom of the total price range. Ways to use the Stochastic Oscillator: – (1) A Buy is indicated when the %K or %D falls below a specified level, typically 30%, and then rises above that level. A Sell is indicated when the line rises above a specified level, typically 70%, and then goes below that level. (2) A Buy is indicated when the %K line rises above the %D line. A sell is indicated when the %K line falls below the %D line. (3) When prices are making new highs and the Stochastic does not exceed its previous highs a divergence occurs, often

indicating a change in the current trend. – The buy/sell signals are triggered when the s%K line crosses the %D line after the %D line has changed direction. At the bottom, the buy signal is generated. At the top, the sell signal is generated. If the Stochastic oscillator is used with intraday charts, some traders suggest charting a weekly Stochastic to check the overall long-term trend on the instrument.

%R - Percent Retracement

%R measures the latest price of an instrument in relation to its price range over a given number of periods. The %R value is plotted on a scale of 0 to -100. This study is similar to the Stochastic, however the scale is reversed. %R values of greater than -20% indicate overbought, and values smaller than -80% indicate oversold. The concepts for interpreting %R is testing the presence of a divergence in the overbought or oversold areas. The suggested time periods are 5, 10, and 20 periods, all based on 1/4, 1/2, or the entire monthly trading cycle. It is not unusual to see an overbought or oversold indication for a long period of time. Then the traders wait for the price to turn up or down before buying or selling or they find themselves in or out of the instrument long before it's price actually moves in the opposite direction.

DMI, ADX, and Directional Indicators

The Directional Movement Index, DMI, provides, on a scale of 0 to 100, an indication of how much movement is present in an instrument, that is, whether the instrument is trending or not. The Average Directional Movement Index, ADX, does the same thing. However, it shows a (smoothed) moving average of the DMI. The higher the ADX, the more the instrument is trending. If an instrument is trending, it becomes a good candidate for trend-following studies. The ADX can also be used to compare several instruments to decide which is trending and which is non-trending, so the trader can take action based on this information. Directional Indicators give buy and sell signals based on upward or downward movement: – (1) The +DI measures positive (or upward) movement. (2) The -DI measures negative (or downward) movement. – A buy signal is indicated when the +DI line crosses over the -DI line. A sell signal is indicated when the -DI line crosses over the +DI line. It should be noted that the buy or sell signals are best taken when the ADX is also indicating a trend.

We have added to the `fSeries` Library a series of trading indicators commonly used in technical analysis. In the following example we list these functions, from which we can easily derive the mathematical formula for the trading indicators. Feel free to add further indicators.

Example: Trading Indicators

The following S-plus functions are part of the `fSeries` Library. The trading indicators are functions of Prices Prices `X` (as `O` Open, `H` High, `L` Low, `C` Close), and `V` trading volume. These functions can be grouped in four major categories. *Utility Functions:* `EMA` Exponential Moving Average, `BIAS` Bias, `ROC` Rate of Change, `OSC` EMA-Oscillator. *Oscillators:* `MOM` Momentum, `MACD` MACD, `CDS` MACD Signal Line, `CDO` MACD Oscillator, `VOHL` High/Low Volatility, `VOR` Volatility Ratio. *Stochastics Indicators:* `FPK` Fast %K, `FPD` Fast %D, `SPD` Slow %D, `APD` Averaged %D, `WPR` Williams %R, `RSI` Relative Strength Index. *Directional Movement Indicators:* `DMP` Plus Directional Movement, `DMM` Minus Directional Movement, `DMX` Directional Movement `ADX` Average Directional Movement.

```
### UTILITY FUNCTIONS: ####

"ema" <- function(x, lambda, startup=0){
      # EXPONENTIAL MOVING AVERAGE:
      # EMA(n) = lambda*X(n) + (1-lambda) * EMA(n-1); lambda = 2 / ( n+1)
      if (lambda >= 1) lambda <- 2/(lambda+1)
      if (startup == 0) startup <- floor(2/lambda)
      if (lambda == 0){ xema <- rep (mean(x),length(x))}
      if (lambda > 0){ xlam <- x * lambda
```

```
          xlam[1] <- mean(x[1:startup])
          xema <- filter(xlam, filter=(1-lambda), method="rec")}
      xema }
"bias" <- function(x, lag) {
      # BIAS: (X - EMA) / EMA
      xema <- ema(x, lag); (x - xema)/xema }
"roc" <- function(x, lag){
      # RATE OF CHANGE INDICATOR: ROC: (X(n) - X(n-k) ) / X(n)
      c(rep(0,times=lag),diff(x, lag=lag)) / x }
"osc" <- function(x, lag1, lag2) {
      # EMA OSCILLATOR INDICATOR: (EMA_LONG - EMA_SHORT) / EMA_SHORT
      xema1 <- ema(x, lag1); xema2 <- ema(x, lag2)
      (xema1 - xema2)/xema2}

### OSCILLATORS: ###

"mom" <- function(x, lag) {
      # MOMENTUM INDICATOR: MOM: X(n) - X(n-lag)
      c(rep(0,times=lag),diff(x,lag=lag)) }
"macd" <- function(x, lag1, lag2) {
      # MA CONVERGENCE DIVERGENCE INDICATOR: MCD: (EMA_LONG - EMA_SHORT)
      ema(x, lag1) - ema(x, lag2) }
"cds" <- function(x, lag1, lag2, lag3) {
      # MACD SIGNAL LINE INDICATOR: SIG: EMA(MCD)
      ema(macd(x,lag1,lag2), lag3) }
"cdo" <- function(x, lag1, lag2, lag3) {
      # MACD OSCILLATOR INDICATOR: CDO: MACD - SIG
      macd(x, lag1, lag2) - cds(x, lag1, lag2, lag3)}
"vohl" <- function(high, low) {
      # HIGH LOW VOLATILITY: VOHL: high - low
      high - low }
"vor" <- function(high, low){
      # VOLATILITY RATIO: VOR: (high-low)/low
      (high - low) / low }

### STOCHASTICS OSCILLATORS: ###

"fpk" <- function(close, high, low, lag) {
      # FAST %K INDICATOR:
      minlag <- function(x, lag) { xm <- x
          for (i in 1:lag){
          x1 <- c(x[1],x[1:(length(x)-1)])
          xm <- pmin(xm,x1); x <- x1}
          xm}
      maxlag <- function(x, lag) { xm <- x
          for (i in 1:lag){
          x1 <- c(x[1],x[1:(length(x)-1)])
          xm <- pmax(xm,x1); x <- x1}
          xm}
      xmin <- minlag(low, lag)
      xmax <- maxlag(high, lag)
      (close - xmin ) / (xmax -xmin) }
"fpd" <- function(close, high, low, lag1, lag2) {
      # FAST %D INDICATOR: EMA OF FAST %K
      ema(fpk(close, high, low, lag1), lag2) }
"spd" <- function(close, high, low, lag1, lag2, lag3) {
      # SLOW %D INDICATOR: EMA OF FAST %D
      ema(fpd(close, high, low, lag1, lag2), lag3) }
"apd" <- function(close, high, low, lag1, lag2, lag3, lag4) {
```

```
      # AVERAGED %D INDICATOR: EMA OF SLOW %D
      ema(spd(close, high, low, lag1, lag2, lag3), lag4) }
"wpr" <- function(close, high, low, lag) {
      # WILLIAMS %R INDICATOR:
      minlag <- function(x, lag){ xm <- x
          for (i in 1:lag){
          x1 <- c(x[1],x[1:(length(x)-1)])
          xm <- pmin(xm,x1); x <- x1}
          xm}
      maxlag <- function(x, lag) { xm <- x
          for (i in 1:lag){
          x1 <- c(x[1],x[1:(length(x)-1)])
          xm <- pmax(xm,x1); x <- x1}
          xm}
      xmin <- minlag(low, lag)
      xmax <- maxlag(high, lag)
      (close - xmin ) / (xmax -xmin) }
"rsi" <- function(close, lag) {
      # RSI - RELATIVE STRENGTH INDEX INDICATOR:
      sumlag <- function(x, lag) { xs <- x
          for (i in 1:lag){
          x1 <- c(x[1],x[1:(length(x)-1)])
          xs <- xs + x1; x <- x1}
          xs}
      close1 <- c(close[1],close[1:(length(close)-1)])
      x <- abs(close - close1); x[close<close1] <- 0
      rsi <- sumlag(x,lag)/sumlag (abs(close-close1),lag)
      rsi[1] <- rsi[2]
      rsi }
```

The investigation of C.C. Yang et al. (1995) makes use of eleven technical indicators, chosen a priori by analysis as inputs for a 11-7-1 connectionist network. Four basic indicators, `ema`, `bias`, `osc`, and `roc` are used as atomic operations to generate the eleven predictors $X_t^{(i)}$. The precise definition of these indicators is given in the following exercise. The response Y_t or output of the system is to reveal the trend of the stock market and to enable us to build a prediction model for assisting in making trading decisions, i.e. the trend of the price movement in the next trading days. Here are the rules:

- If $Y_{t-1} >$ lower threshold of SELL and $Y_t >$ upper threshold of SELL then SELL one unit,
- if $Y_{t-1} <$ upper threshold of BUY and $Y_t >$ lower threshold of BUY then BUY one unit,
- otherwise HOLD.

with the following two constraints:

- If there have been two consecutive SELLs and the closing PRICE is still increasing then HOLD until there is a significant withdrawal in the PRICE (e.g. 10%),
- if there have been two consecutive BUYs and the closing PRICE is still decreasing then HOLD until there is a significant return in the PRICE (e.g. 10%).

The upper and lower thresholds for the SELL signal were set to 0.9 and 0.6, respectively, whereas those for the BUY signal were set to 0.1 and 0.4. The combination of the upper and lower thresholds enables the system to smooth and those unwanted noisy surges.

Exercise: Trading the Pacific Basin Capital Markets

The predictors and the response in the notation of C.C. Yang et al. are

- $X_1 = K_n^{12}$
- $X_2 = K_n^{12} - D_n^{12}$
- $X_3 = RSI_n^6$
- $X_4 = RSI_n^6 - RSI_n^{12}$
- $X_5 = MA_3(FR_6)$
- $X_6 = OSC_{3,10}(FR_6)$
- $X_7 = VA_6(MA_3(RSI_n^6))$
- $X_8 = VA_6(MA_3(RSI_n^6)) - VA_6(MA_3(RSI_n^{12}))$
- $X_9 = OSC_{3,6(C_n)}$
- $X_{10} = BIAS_10(C_n)$
- $X_{11} = BIAS_10(V_n)$
- $Y = FK_n^{j,k}$

where K is defined by the function `fpk()`, D by `fpd()`, RSI by `rsi()`, MA by `ema()`, OSC by `osc()`, and $BIAS$ by `bias()`. Additionally FR is defined by

$$FR_n = \frac{H_n - L_n}{C_{n-1}} \ ,$$

and VA by

$$VA_n = VA_{n-1} + \frac{2C_n - L_n - H_n}{H_n - L_n} V_n \ .$$

The predictive trend FK is calculated as

$$FK_n^{j,k} = \frac{C_{n+1} - MIN_{i=n+1+k}^{n+1+k}(C_i)}{MAX_{i=n+1+k}^{n+1+k}(C_i) - MIN_{i=n+1+k}^{n+1+k}(C_i)},$$

where as usual C, H, L, and V are Close, High, Low, and Volume, respectively.

The `fSeries` Library provides data for the Hang Seng (Hong Kong), TSE (Taiwan), and Nikkei (Japan) Index. For missing data (beside for Saturday and Sundays) add the price and volume data from the previous business day. Thus a trading decision for those missing days does not add to a positive or negative return. Then calculate the 11 predictors X and the response Y. Perform the investigation in form of a moving window on a training set of 1 year or 250 business days, and a forecasting set of approximately 1 month, i.e. 20 days. Produce charts for the index, the returns, the volume, and the trading indicators over the whole set of data. Then we explore for the returns the basic statistical properties moving along the data set. Finally simulate the trading results.

Notes and Comments

Linear and Generalized Linear Models: There are several statistical textbooks around concerned with the LM and GLM approach. These include for example the books of McCullagh and Nelder, *Generalized Linear Models*, (1989), and Dobson, *An Introduction To Generalized Linear Models*, (1990). Beside these textbooks several tutorials are available on the Internet. We followed partly the course material written by Rodriguez, *Linear Models for Continuous Data* (Chapter 2), and *Generalized Linear Model Theory* (Appendix B), (2001). Regression models with discrete variables are described in the lecture notes of Fox, *Logit and Probit Models*, (2001). A very helpful source describing both the theoretical aspects of Linear Modelling and the Splus software is Venables' and Ripley's (2001) book *Modern Applied Statistics with S*, Chapter 7 on *Generalized Linear Models*.

Additive and Generalized Additive Models: GAMs are described in the textbook *Generalized Additive Models* written by Hastie and Tibshiranie (1995). The section presented here on GAM's is mainly based on Chapter 2 and Appendix B of the lecture notes of N.N. (2001) and publications of the SAS Institute, ... ().

Projection Pursuit and Generalized Projection Pursuit Regression: This section followed the review article of Klinke and Grassmann (1998). What follows is a list of further references taken from this review: The idea of "Projection Pursuit" has been introduced by Kruskal (1969; 1972) for exploratory data analysis. The approach has been successfully implemented for exploratory purposes by many authors, e.g. Friedman and Tukey (1974), Jee (1985), Huber (1985), Jones and Sibson (1987), Friedman (1987), Hall (1989), Cook and Cabrera (1992), Cook, Buja and Cabrera (1993), Posse (1995), and Nason (1995). The idea has been applied to location and symmetry estimation by Blough (1989) and by Maller (1989), to density estimation by Friedman, Stuetzle and Schroeder (1984), by Chen (1987), Rejtö and Walter (1990, 1992), to classification by Friedman and Stuetzle (1981a) by Portier, Dippon and Hetrick, (1987), by Flick, Jones, Priest and Herman (1990), and to discriminant analysis by Posse (1992), by Ahn and Rhee (1992), and by Polzehl (1995). Good references about projection pursuit are Jones and Sibson (1987) and Huber (1985). *Generalized Projection Pursuit Regression*, GPPR, is a natural extension of PPR to exponential family distributions. For further information we refer to the paper *Generalized Projection Pursuit Regression*, written by Lingjaerde and Liestol (1998) and to the paper *Logic Response Projection Pursuit* written by Roosen and Hastie (2000).

Multivariate Adaptive Regression Splines: The material presented for the MARS methodology was taken from Jerome Friedman's paper *Multivariate Adaptive Regression splines* and the material about POLYMARS was taken from a series of publications written by Kooperberg, Stone and coworkers. Other references to the MARS methodology include the papers ...

Other Regression Models: Note, that the models fitted by GAM and PPR are examples of nonparametric regression. S-PLUS includes other functions for performing nonparametric regression, including the `ace()` function, which implements a technique for nonparametric regression, *alternating conditional expectations*, or `avas()`, *additive and variance stabilizing transformations*. For details concerned with these approaches we refer to Chapter 8, *More on Nonparametric Regression* of the *Splus Guide to Statistics* (1999).

2.4 Neural Networks: Feedforward Connectionist Networks

Introduction

Neural Networks have attracted during the last years so many attention, especially in the finance community, so that we devote an extra section to this issue.

Inspired more from biological aspects than statistical concepts neural networks were becoming probably one of the most prominent tools for parametric non-linear time series analysis and forecasting. Its close relationship to regression analysis allows to use the networks for the analysis and forecasts of financial markets. The research literature is huge on these topics and today the basic material is also available in many textbooks about neural networks.

In the following we concentrate on the neural network approach developed in the context of modelling economic and financial market data in the 90ties. We introduce neural networks of the "feedforward connectionist network" and present some efficient modelling schemes to make the networks available for forecasting and trading purposes.

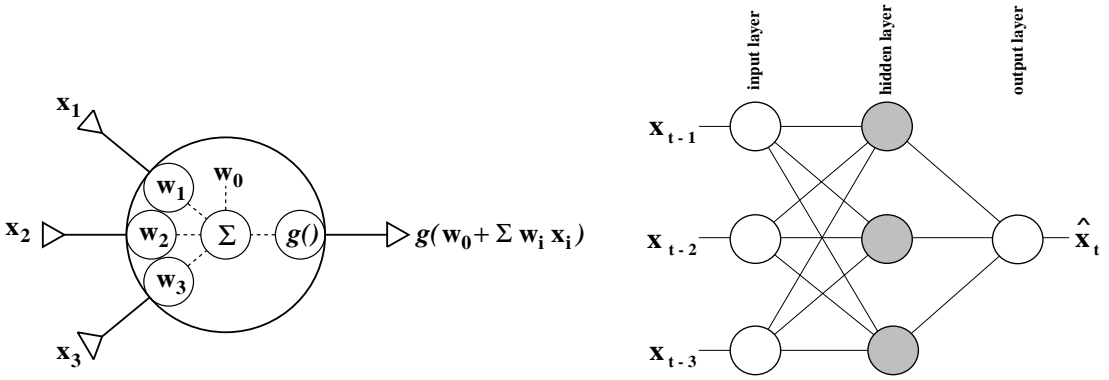
2.4.1 Regression by Neural Networks

Neural networks of the "feedforward" type represent a certain class of nonlinear functions $f : \mathbb{R}^m \to \mathbb{R}^n$. For time series analysis the output space is usually one-dimensional ($n = 1$). We specify the network to read

$$\widehat{x}_t = f(x_{t-1}, \ldots, x_{t-I}) = w_0 + \sum_{l=1}^{H} w_l \tanh\left(a_{l0} + \sum_{k=1}^{I} a_{lk} x_{t-k}\right). \tag{2.121}$$

Here we have written I for "input" and introduced another net parameter H for "hidden". This equation defines a "single hidden layer feedforward connectionist network". Figure 2.3.1 shows a sketch of such a network ($I = 3$, $H = 3$) together with a prototype node. The value of the function is denoted by $\widehat{x}_t$, to note that the output is an "estimate" of the time series at time t based on the I immediately preceding values. Thus the net can serve to model the process $x_t = f(x_{t-1}, \ldots, x_{t-I}) + \epsilon_t$. The term "feedforward" characterizes the structure of the net determined by the connections.

However, the network is not restricted to the form as written in equation 2.121. It can be used in a much broader sense



■ Figure 2.5.1 Prototype node and connectionist network configuration for the purpose of time series analysis. The node computes a weighted (w_i) sum of the input values (x_i), adds a threshold constant (w_0) and applies the transfer function ($g()$ (linear, tanh() or another choice). The output of the node os therefore $g(w_0 + \Sigma w_i x_i)$. The bottom figure shows a "single hidden layer feedforward connectionist network" with three units in the input and hidden layer. The output layer consists of one node only. The shading denotes nonlinear transfer functions. The empty nodes apply the identity as transfer function. This is a graphical representation of equation 2.121 with $I = 3$ and $H = 3$. *Source: deGroot, (1993)*

$$\mathcal{Y}_t = f(\mathcal{X}_{t-1}, \ldots, \mathcal{X}_{t-I}) = w_0 + \sum_{l=1}^{H} w_l \tanh\left(a_{l0} + \sum_{k=1}^{I} a_{lk}\mathcal{X}_{t-k}\right), \tag{2.122}$$

where we perform a regression type analysis of predictors $\mathcal{X}$ against one or more response values $\mathcal{Y}$.

There is an *input layer* with I nodes, a *hidden layer* with H nodes and an *output layer* with (usually) a single node. *Nodes* are computing elements of these nets, each has several input connections and one connection for the output. Another term for nodes is "units". The output is computed based on the "activity" of the node by applying the node's *transfer function* g. In our networks, the input and output layer consist of nodes with $g(x) = x$, while the nodes of the hidden layer show nonlinear transfer characteristics, e.g. modelled by $g(x) = \tanh(x)$. The *activity* y of each node is computed via a weighted (w_i) sum of its inputs (x_i) and an additive threshold (w_0) to be $y = w_0 + \sum_i w_i x_i$. All w_i and thresholds of a net are called *weights* for simplicity. Therefore in equation (2.121) the weights are the elements of $\{w_0, w_l, a_{l0}, a_{lk}\}, l = 1, \ldots, H; k = 1, \ldots, I$. There are two types of parameters in equation (2.121). The first type determines the *network topology* (I,H), while the other type fixes the mapping (weights). In other words: Of all smooth mappings $f : \mathbb{R}^I \to \mathbb{R}$, equation (2.121) defines a subclass, which is a certain "feedforward artificial neural network with one hidden layer". This subclass of mappings is "parameterized". Fixing the number of units in the hidden layer means further reduction of the subclass of functions. Specification of weights selects one of the elements of the reduced subclass.

In what sense does the network implement a function approximator?

A complete specification of net topology and weights fixes the mapping realized by the network. The approximative nature of this mapping is defined with reference to a "desired" mapping. In artificial neural network applications this mapping is known only for a set of values ("examples"), a closed form is unknown. Specifically, there is a set of input values $\{x_p\}_{1 \le p \le P}$, $x_p \in \mathbb{R}^I$ (called

"input patterns") and for each of these there is a corresponding "desired" (or "target") pattern z_p, $p = 1, \ldots, P$. Patterns come in pairs (x_p, z_p). The approximation realized by the network is expressed by means of an "error functional" E which takes all patterns into account:

$$E = \frac{1}{2} \sum_{p=1}^{P} (f(x_p) - z_p)^2 \tag{2.123}$$

This error functional has to be minimized, i.e. the "desired" mapping is approximated by the network on the basis of the "examples". The functional value $f(x_p)$ computed by the network, when pattern x_p is "presented" at its inputs, is usually called "obtained" pattern. The process of estimating network parameters (weights) with respect to this error functional raises a nonlinear optimization problem. Solving this iteratively is called "learning", the algorithm is called "learning algorithm". The most widely used algorithm is "backpropagator" as described in Rumelhart and McClelland (1986).

2.4.2 Time Series Analysis With CNAR Models

For a successful time series analysis approach with connectionist networks several aspects have to be considered. They include usually a process of preprocessing the data, a scheme to select pattern, an initialization process for the network parameters, the estimation of the network parameters itself together with a monitoring process, diagnostic tools, and forecasting tools together with a combination approach of individual networks.

Neural Networks and Dynamical Systems Analysis

Neural networks can be utilized for dynamical systems analysis. Consider a nonlinear dynamical system evolving in a space of small dimension m. The time evolution of the system is described in continuous time by

$$\dot{x}(t) = F_\mu\left(x(t)\right), \quad x \in \mathbb{R}^m. \tag{2.124}$$

The set of coordinates describing the system is denoted by x, F_μ determines the nonlinear time evolution of coordinates, μ serves as (experimental) control parameter. In discrete time, the evolution can be formulated to read

$$x(n+1) = f_\mu\left(x(n)\right), \quad n \in \mathbb{N}. \tag{2.125}$$

In the last two decades those dynamical systems have extensively been investigated which show a regular behavior for small values of the control parameter, but whose trajectories become more and more complicated as μ is increased. Special attention has been paid to dissipative systems exhibiting "chaos". A linear analysis models the behavior of the system by introducing modes (e.g. independent oscillators). Increasing μ leads to an increased number of excited modes: the more excited modes, the more complex the (dissipative) system. In this interpretation a continuous power spectrum of the dynamical system indicates an infinite number of excited modes. Introduction of nonlinear dynamics allows a different interpretation of such a spectrum: The spectrum may be generated by a system evolving nonlinearly in finite dimensions. In the nonlinear context the linear dimension concept of "number of excited modes" has to be replaced by new concepts like "number of non-negative characteristic exponents" or "information dimension". For a detailed introduction to dynamical systems analysis we refer to Eckmann and Ruelle (1985).

For the task of reconstructing the dynamics from an experimental signal of the system it is customary to use time delays. A m-dimensional signal $x(t)$ from scalar measurements $u(t)$ is obtained by choosing different delays $T_1 = 0, T_2, \ldots, T_N$ and writing $x_k(t) = u(t + T_k)$. In this manner a N-dimensional signal is generated. This results in a projection of the trajectories of the system in N dimensions. Depending on the choice of variable u and in particular on the time delays, the projection will look different. These choices for the reconstruction of a dynamical system have to be made carefully. An important observation in this context needs to be discussed briefly. Measurements on a physical system supply information of limited precision due to a number of independent effects. First of all the setup of an experimental situation may be hard to reproduce with the required degree of accuracy. This is important for systems

with sensitive dependence of their time evolution on initial parameters. Furthermore there are measurement errors. These limit the information as well.

There are a number of properties of neural networks which make them attractive for an application in the context of reconstructing dynamical systems. Since the data to construct the model from is a time series, patterns can be formed in the same way as using time delays. A pattern is formed from the time series $\{x_t\}$ by specifying a "grouping scheme" $\tau_1, \ldots, \tau_I$ with respect to a certain time index t. The input pattern then reads $(x_{t-\tau_1}, \ldots, x_{t-\tau_I})$, the desired output will be x_t for example. Since the transfer function of the units in the hidden layer is a hyperbolic tangent, there is nonlinearity in the network which may allow to approximate complicated time series, even with chaotic behavior. The ability of networks to deal with data corrupted by noise, is an interesting feature. This might enable the network to extract the correct dynamical model from noise corrupted measurements of limited precision. Another feature of neural network is their adaptiveness. Since the model parameters are determined via an optimization procedure, the mapping is not completely determined by the net topology. For example, if some of the connections are reduced to zero, the mapping may change its overall characteristics. Another motivation for applying neural networks to reconstruction tasks are theoretical results showing that these networks are universal function approximators, Cybenko (1988), Funahashi (1989), Hornik, Stinchcombe and White (1989). In other words, our networks are capable of arbitrarily accurate approximation to any real-valued continuous function over a compact set.

With respect to the problem of reconstructing dynamical systems, Lapedes and Farber (1987) were among the first researchers to perform an empirical study. They explored the ability of neural networks on nonlinear signal processing tasks. One of their examples was the logistic map $x_{t+1} = \mu x_t(1 - x_t)$. This map shows many of the properties described above for dynamical systems. Here, μ serves as control parameter. For μ greater than a certain constant ($\mu_c \approx 3.5699$) the system shows chaotic behavior for almost all choices of μ. This results in a continuous and flat power spectrum, although the system is one-dimensional and deterministic. Lapedes' and Farber's goal was to adjust the weights in a network, enabling a prediction of the next point x_{t+1} in this chaotic series ($\mu = 4.0$) given the present point x_t. The network structure of their investigations was specified with one input, five hidden, and one output units. The input and output units had a linear transfer function, while the five hidden units were assigned sigmoid ones $g(x) = 1/2\,(1 + \tanh(x))$. Lapedes and Farber also provided a direct connection from the input node to the output node. They initialized the network with small random weights and then "trained" the system with backpropagation on a sample of 1000 pairs $(x_t, xt + 1)$. Prediction was performed on 500 different pairs with a normalized root mean square error of $1.4 \cdot 10^{-4}$. They also tested the approximation by iterating predictions ("prediction on predicted values"), i.e. the output of the net from the previous step serves as the net's input in the next step. This worked well for a number of iterates. This early work of Lapedes and Farber had a strong resonance among physicists working in the field of nonlinear systems and chaotic time series. Casdagli (1989) published a comparison of different methods on the problem of constructing a "predictive model directly from time series data" [Cas89]. In this paper Casdagli summarized some algorithms developed for this task and compared their abilities by applying them to a choice of chaotic time series. Among the conclusions of this comparison was a "superiority" of the neural network technique. Casdagli also explicitly mentioned the close relationship between reconstruction tasks and statistical time series analysis.

Data Preprocessing

In the preprocessing of data we consider two aspects: 1) Standardization of pattern and 2) the proper selection of predictors.

Pattern Standardization

There are a number of "shifting" and "scaling" operations reported in the literature. When the network has a linear output node, the estimated value is not restricted to a fixed interval. In principle there is no need for scaling, and the network is able to model a wide range of values. In order to handle data from a variety of sources we perform *data standardization* known from statistics to our data: At each input node k the mean $\langle x_{t-k} \rangle$ and the standard deviation $\sigma_{x_{t-k}}$ of all N input patterns are computed:

$$\begin{aligned} \langle x_{t-k} \rangle &= \frac{1}{N} \sum_p x_{t-k}^{(p)}, \\ \sigma_{x_{t-k}} &= \frac{1}{N-1} (\sum_p (x_{t-k}^{(p)} - \langle x_{t-k} \rangle)^2)^{1/2}. \end{aligned}$$

Then the value $x_{t-k}^{(p)}$ of the p-th pattern at this node transforms into

$$\tilde{x}_{t-k}^{(p)} = \frac{(x_{t-k}^{(p)} - \langle x_{t-k} \rangle)}{\sigma_{x_{t-k}}}. \tag{2.126}$$

Standardization scales the data to a common numerical range, which proves especially useful in multivariate analysis.

Indicator Selection

Another important step of data preprocessing is the identification of input variables. In univariate modelling this means specification of time lags; however, multivariate modelling requires identification of relevant predictors. In the univariate case one can compute the partial autocorrelation function, PACF, to get some hints on significant lags. Multivariate modelling is more difficult than univariate. Usually, there exist a large number of potential predictors as possible input variables for a network. A quite common approach for the search on predictors is based on plausibility considerations. Another approach is based on statistical tests. Each selected predictor value is paired with the response value. The tests then evaluate the significance level of "correlations" among data. The outcome of the test procedures leads to an ordering of the time series with respect to their forecast quality.

Pattern Selection

Random selection of validation examples for the optimization is reported about in the literature. Authors make use of the validation set to decide when to stop optimization, i.e. the value of the performance criterion is followed for both sets during learning epochs. In order to eliminate the random nature of the decision, which pattern to assign to which set, we need an interpretation of the validation idea. After that we need a deterministic algorithm based on this idea. This raises the question of how reliable the net estimates are in regions where no training patterns

are available. If any of the validation patterns are close to training patterns and if we choose the correct model, then the residuals of the estimated values and the validation patterns must be similar to those of the training set. Such a smooth mapping is desired in order to keep the risk of overfitting small. "Similar" is meant with respect to the mean and the variance of the training residuals. If, however, the validation patterns are far away from patterns in the training set, nothing particular can be said. What does "close" and "far" mean? These terms are meaningful, if they are related to the density of training patterns. This density is determined on the input patterns in $\mathbb{R}^I$ if there are I input units. It would also be advantageous to incorporate the density of patterns into the performance measure. The current formulation of the performance measure favors accurate modelling where patterns are dense, while other less dense regions can be approximated with less accuracy. However, if we wish to model phase space trajectories, we have to model seldomly visited points as accurately as often visited ones, which could be achieved by the introduction of weights for the patterns. Patterns which belong to high-density regions have a small weight, patterns in sparse regions carry a larger weight. The performance criterion would then read

$$E_W = \frac{1}{2} \frac{\sum_p W_p (\widehat{x}_p - x_p)^2}{\sum_p W_p}, \tag{2.127}$$

where W_p denotes the weight of pattern p. The question then arises how to determine these weights in a meaningful and non-arbitrary way.

deGroot and Würtz (1991) proposed a clustering algorithm for both tasks, the selection of training/validation patterns and weight assignment. Here, "clusters" are defined as in the k-means problem described by Hartigan and Wong (1979). The cluster centers are identified as "representative patterns" for a certain region. The number of patterns per cluster represents the density of patterns in that region. If there are N patterns available in total and we decide the training set to be of size T, then there are $N - T$ patterns in the validation set. We run a clustering algorithm to find T clusters, with the conditions that the cluster centers are patterns and the totaled sum of the variances in the clusters is as small as possible. If a cluster has exactly two members we randomly select one of them to be the center. Distances in the clustering process are Euclidean. Only input coordinates are considered when computing the distances. The reason for this latter requirement is that the similarity criterion must exclude the target value, because the target value is believed to be the true value plus an error. Therefore, if the inputs coincide but the target values of two patterns differ, this discrepancy is solely due to the error term. After the clustering we transfer the cluster centers to the training set, while the other patterns are transferred to the validation set. The validation patterns get a "weight". This is the inverse of the cardinality of validation patterns belonging to the same cluster. Example: If there are six validation patterns in a cluster, then each of them gets a weight of 1/6.

The cluster algorithm operates in a deterministic manner. Initially each pattern belongs to a cluster of its own. Then the closest two patterns are joined into one cluster. After that the distance matrix is updated with respect to the new cluster center. Then again the two nearest clusters are joined and so on. Each joining operation reduces the number of clusters by one. However, for a good solution we need an algorithmic step which transfers single patterns from one cluster to another, if that reduces the total variance. For this reason after each joining operation the patterns are checked whether they can be transferred to another cluster with a reduction of variance. The algorithm terminates if the number of clusters equals T. This procedure works well. However, the problem how to determine the appropriate number of training patterns needs

to be solved. A large number of training patterns leads to small clusters. As soon as there is only one member in the final cluster, there is no validation value. On the other hand, we would like both sets to be similar to each other with respect to the monitored values described below. This requires T not to be chosen too large.

Initialization of Network Parameters

If we insert the approximation $\tanh(x) \approx x$ into equation 2.121 we get

$$\widehat{\tilde{x}}_t = \left(w_0 + \sum_{l=1}^{H} w_l a_{l0} \right) + \sum_{k=1}^{I} \left(\sum_{l=1}^{H} w_l a_{lk} \right) \tilde{x}_{t-k}. \tag{2.128}$$

This approximation holds for small arguments of the hyperbolic tangent. In equation 2.128 we have rearranged the terms of the sum in order to obtain similarity with the notation in equation (2.5). In this limiting case the network behaves like an AR-model of order I. The conditions for the "degree" of the AR behavior are specified by the activities in the hidden layer. The dynamics of these activities is governed by an inner product of patterns and weights: The activity y of node l is computed as $y_l = a_{l0} + \sum_k a_{lk} \tilde{x}_{t-k}$, where a denotes constants. We discuss the role of the weights first. If the weights at a specific hidden node are small, then that node has essentially a linear transfer characteristic. From this observation we draw a first conclusion. Our connectionist approach, which is "justified" by its nonlinearity, includes the potential to operate like a linear network, i.e. in our context a linear time series model. What does the word "potential" mean in this sentence? We have to keep in mind that the final model will be a result of a nonlinear optimization procedure. Thus, if the optimization algorithm converges with small weights at the hidden layer, then the linear potential of the network was realized by the optimization process. The key to our connectionist methodology lies in proceeding in the reverse direction. If we initialize the network with small weights at each hidden unit, then we know that we have an approximately linear model. Starting from this linear initialization the optimization algorithm will then be able to increase the activities at the hidden units, thereby enlarging the arguments of the hyperbolic tangent, i.e. increasing the nonlinearity of the time series model. So far we have neglected the contributions of the patterns to the activities of the hidden nodes. If we compute the weights in our initialization procedure we have to make sure that the patterns do not destroy the validity of our linear approximation. Here we benefit from our standardization of input variables, because this procedure essentially confines the numerical values of the input patterns to a range of approximately ± 3.

Hidden Layer Initialization - PCA

Our goal is now to create an initial linear model such that equation 2.128 is a valid approximation to the nonlinear equation. The simplest realization would be an initialization of the parameters with small random numbers. We will compute instead a full linear model as the initial guess of the parameters, because this provides additional knowledge about the model properties which turns out to be useful. Since there are more parameters in the connectionist network than in the AR-model, we need more equations. From equation (2.1) we deduce that each hidden unit performs a projection of the input pattern vector on an axis given by the weights at that node. Now suppose the data to be presented to the network show internal dependencies (e.g. linear correlations). This may be the case for univariate as well as multivariate data. If we wish to

process "informative" data only, we have to consider these dependencies. In order to take linear correlations among our input data explicitly into account, we write the input X to the net as a linear combination of some input data vectors Y_k:

$$X = \alpha_1 Y_1 + \alpha_2 Y_2 + \cdots + \alpha_I Y_I. \tag{2.129}$$

Now Y_1 is supposed to incorporate as much information of the inputs as possible: Y_1 may also be written as a linear combination of the $\tilde{x}_{t-k}$. In this context "information" should be read as "maximum variance". If we project the data onto the axis Y_1, the squared sum of the residual vectors should be minimal. By residual vector we mean the difference between the original and the projected vector. In the next term of equation (4.3), Y_2 is calculated the same way. This time we have the additional requisition that Y_2 must be orthogonal to Y_1. This orthogonalization procedure is continued up to Y_I. In fact, this recipe for extracting relevant information from data is well known in statistical data analysis under the name *principal component analysis* (PCA). In a practical application the principal components (which we called Y_k) are computed simultaneously. They are the eigenvectors of the correlation matrix of the standardized data. The corresponding eigenvalues represent weights imposed on the eigenvectors. The larger the eigenvalue the more important the component. In statistics the eigenvalues are utilized to distinguish "real" information from noise. There are a number of criteria to do so. Since the sum of the eigenvalues (λ_i) equals the trace of the correlation matrix, one selects only those eigenvalues which are larger than a certain threshold, which is chosen to be not much smaller than one (e.g. $\lambda_i > 0.8$). Another common choice is to incorporate all largest components, until their sum exceeds a certain fraction $(1 - \alpha)$ of the trace. (With $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_I$ select the smallest element of $\{n : \sum_{i=1}^{n} \lambda_i \geq (1 - \alpha) \cdot \text{trace}\}$.) The disadvantages of the method may be seen in their derivation from arguments valid only in the linear case and the open question of how to initialize additional hidden units if there are more of them than there are input units. The advantages, on the other hand, may be seen in their conceptual simplicity, their clear interpretation and their data based derivation. Here again we "reverse" the approach of the literature. There are a number of authors demonstrating that the final result of the optimization can be the space spanned by the principal components, see e.g. Oja (1982), Baldi and Hornik (1989), Malki and Moghaddamjoo (1991). Instead we initialize the network to have this property, because we are interested in the additional (nonlinear) capabilities of the network. We summarize the first steps of weight initialization in the following algorithm.

Algorithm: Weight Initialization

- At each input node k standardize the data of pattern p (and the target values).
  $\tilde{x}_{t-k}^{(p)} = (x_{t-k}^{(p)} - \langle x_{t-k} \rangle)/\sigma_{x_{t-k}}$.
- Compute the correlation matrix C of the standardized data.
  $(C)_{kl} = \sum_p \tilde{x}_{p-k}\, \tilde{x}_{p-l}/(N-1)$
- Compute eigenvalues λ_j and normalized eigenvectors n_j of C.
- Standardize eigenvectors. $t_j = \sqrt{\lambda_j}\, n_j$

The last step is intended to reflect the importance of the component in the weights. The I components of the eigenvector t_l (C is an $I \times I$ matrix) are now the weights at the hidden unit l, i.e. $a_{lk} \equiv (t_l)_k, k = 1, \ldots, I; \quad l = 1, \ldots, H$. If we look at equation 2.121 we find that this construction leads to the desired projection of the input vector onto this eigenvector. Since we standardized the data we simply put $a_{l0} \equiv 0$ for all l.

Output Layer Initialization - AR Modelling

We are now ready for estimating the weights at the output layer. This initialization follows the linear methodology, because we wish to initialize the net to approximate a full linear AR-model of order I. We therefore compute the weights via equation 2.128 and a least squares fit (w_0 should be close to zero as well due to standardization). *Thus we have computed a complete initial solution of the linear network.* This computation provides an estimate for each parameter. No random numbers are involved in this procedure. On the other hand, it is not a unique estimate, because eigenvectors are only defined up to a reversal of sign. However, this property does not change the linear solution, because the AR-modelling procedure takes care of it.

Now we have to return to the question of validity of the linear approximation. This means we have to discuss how to make use of the initial estimate. If we want the result of our linear calculations to be of any value for the net initialization, we have to concentrate on the weighted sum at the hidden units, because these are the arguments of the hyperbolic tangent: $\sum_k a_{lk}\tilde{x}_{t-k}$ (recall that $a_{l0} \equiv 0$). Since the a_{lk} are fixed, we have to look at the patterns. Among all the projections at the hidden nodes there will be a maximum for a particular pattern. We determine the largest absolute value of this sum (projection) by presenting all the learning patterns to the net. This "critical" value requires the validity of the linear approximation $\tanh(x) \approx x$ as the "worst" case. In most situations it will be sufficient to look at the first hidden unit, whose weights correspond to t_1, since this is the t_j of greatest length by construction. As we are still considering the linear net, we are free to reduce this maximal projection to any desired small value by an appropriate scaling factor without altering the general approximation capabilities of the network, i.e. the AR-model. We call this scaling factor γ. It controls the limit of linearity of our initialization. Thus, our method requires the introduction of one additional parameter. Generally, additional parameters introduced by any method must be considered a weakness of that method. However, our parameterized construction seems to be acceptable, because we can give a clear meaning to γ: we measure the degree of nonlinearity of our initial weight configuration by this γ. This parameter is also fixed at the very beginning of and unaltered during the analysis. — We still have to decide how to compensate the scaling introduced by γ. We choose to scale the patterns appropriately, the alternative being a decrease of the weights at the hidden layer. Rumelhart (1988) observed and postulated that the weights of the network have to be small. If we decrease the weights at the hidden layer, this has to be compensated by enlarging the weights at the output layer (equation (4.2)), thereby violating Rumelhart's rule. For this reason we decided to scale the patterns. This will also change the threshold of the output node, but since this value is already small and will only be decreased, scaling of patterns is the better alternative. Again we summarize the technical steps of the linear initialization.

Algorithm: Linear Initialization:

- Compute the maximal projection y'_{max} of all learning patterns
  ($p \in L$) on the t_j.
  $y'_{max} = \max_{p \in L} \{ |\sum_k a_{jk}\tilde{x}_{p-k}|, j = 1, \ldots, H \}$
- Choose the "degree of non-linearity" γ (typically $0.05 \leq \gamma \leq 0.5$).
- Scale the patterns such that maximal projection equals γ.
  $\tilde{x}'_{p-k} = \tilde{x}_{p-k} \cdot \gamma / y'_{max} \quad \forall p \forall k$
- Compute the weights at the output layer with respect to the outputs of the hidden layer via the AR-modelling procedure.

All these algorithmic steps are carried out for the linear network, i.e. equation 2.121 without the hyperbolic tangent.[3]

Benefits from the Linear Initialization:

At this point we have an initial estimate of the network parameters and a measure for the validity of the linear model. The parameters specify a linear model of order I. What are the benefits of this situation? Why is it advantageous to compute the initial estimate rather than initialize the weights randomly? The advantage of our procedure is that we have complete knowledge of the situation. The "zero-th" iteration of the optimization is well understood. We know the degree of linearity and nonlinearity of our model. We can give arguments why we chose the particular number of hidden units. We can rank the hidden units according to their importance. We know that the projections at the hidden units are in mutually orthogonal directions. And we have the performance of the linear model as a yardstick for our nonlinear estimation process.

Our linear initialization allows further steps to be taken. We may use techniques from linear modelling to judge the relevance of input variables. This is interesting for both, univariate and multivariate situations. If we find (linear) correlations between input variables, we know that PCA will detect them. For this reason, dependencies among the input variables are immediately reflected by the net structure. On the other hand, choosing (linearly) independent variables will result in more hidden units which are also more significant, providing greater modelling power to the network. It is a common standpoint in the neural network community that the network is able to "extract" relevant and to neglect irrelevant data during the optimization process. For this reason, there is no special treatment of irrelevant data, e.g. a removal from the training set. Our method favors a different attitude. Every additional (hidden) unit introduces a number of additional parameters. Since the network can be very sensitive to misspecified models, these parameters are likely to fit the noise rather than being reduced to zero. For this reason we prefer preprocessing of data leading to less parameters for the network rather than the opposite, which is common in the neural network literature.

Parameter Estimation

We now turn to the process of parameter estimation. This must be achieved numerically by an optimization algorithm.

Training the Net with Higher Order Optimization Algorithms

The presented connectionist method provides a linear approximative model as initial estimate. The optimization algorithm now may explore the nonlinearity of the patterns and modify the weights appropriately. This includes decreasing the weights at the hidden units, leading to an even more linear model. We are therefore interested in a procedure that will examine the "surroundings" of our starting point and then proceed in the appropriate direction.

deGroot and Würtz (1991) have presented results which demonstrate the superiority of a second order optimization algorithm for the estimation process of the network parameters. The initial-

[3]We would like to mention that the implementation of this initialization procedure sounds more costly than it really is when actually coded. Mostly elementary numerical procedures of linear algebra are required. These are readily available in public domain libraries like NETLIB or STATLIB.

ization allows further justification of higher order approaches. Since we compute the parameters of the initial solution via the AR-modelling procedure to be a "linear" minimum, we expect a small gradient (depending on γ) in the nonlinear equation as well. In fact, if we choose γ small enough, the optimization algorithm will not be able to distinguish the linear from the nonlinear situation, because of numerical imprecision. Let us have a look at the characteristics of backpropagation in this case. If the chosen step size is too large, then we will be off our initial estimation after just one single step. If, however, we choose a very small step size, we will not be able to move away from the initial estimate within a reasonable amount of time. Therefore we need an optimization algorithm that takes the local structure at our starting point adaptively into account.

A comparison of various optimization algorithms of different orders has been performed by deGroot and Würtz (1991). The conclusions are: As long as computer memory does not enforce limitations a second order optimization algorithm is preferable. The numerical results presented in the following were obtained with the NETLIB implementation of the BFGS algorithm. The most prominent benefit from choosing a second order algorithm is the short execution time. The introduction of these algorithms reduces "learning" to the order of minutes on a PC or Unix based workstation. Thus CNAR time series analysis can be performed interactively.

Monitoring the Parameter Estimation Process

We may also ask whether or to what extent we can make use of our knowledge about the properties of the initial solution. It turns out that we can extract valuable information about the nonlinear solution, if we "monitor" characteristic values of the current solution during optimization. We have already met an example of monitoring when we explained the performance criterion for the training and validation set, respectively. The monitoring information allows an interpretation of what is going on in the network at least on a qualitative level. Before we describe the actual values we monitor in the next paragraphs, we wish to discuss the potential benefits briefly. Some of the values we will monitor were introduced by Weigend and Rumelhart (1990). However, since they started from a randomly initialized set of parameters, they were only interested in the structures developing during optimization. In contrast, we pay attention to changes in these characteristic values. We are then able to interpret these changes qualitatively, because we have our initial model with its known properties included in the process. If we are able to identify and monitor values which are related to the overall net properties rather than to the particular set of patterns, we have a chance to observe the global net behavior. But more information related to the pattern set is also desirable, because we can hope to augment our knowledge from the performance criterion with this additional data. Although the *performance criterion* does not inform too well about inner changes of the model it is nevertheless an evident value to "monitor". It is interesting to relate the internal development of the net to the behavior of the general performance.

Regression Analysis - Linear versus Nonlinear Net

An interesting fact is the usefulness of the nonlinearity of the network. Since we know that the net has the potential of linear modelling, we would like to be sure that it operates successfully in the nonlinear regime. Otherwise we could be tempted to drop the whole network method and apply linear modelling. If we compare the linear approximation of the network equation 2.128 to the real network 2.121, we see that if the net is essentially linear the outputs of both

"networks" (i.e. equations) should not be too different from each other. We therefore propagate each pattern through both networks with identical parameters. (The nets differ only in the presence or absence of the hyperbolic tangent.) This results in a pair of output values for each input pattern. We denote the output of the linear (nonlinear) net for pattern p by η_p (ζ_p). We plot the linear versus the nonlinear output for each pair and fit a linear regression to this set of points. If the network were completely linear, all points should lie on the diagonal resulting in a slope of one. For this reason we monitor the *slope of the regression line* together with its estimated error. Let the regression $\zeta(\eta) = a + b\eta$, then we estimate

$$\begin{aligned}
\langle \eta \rangle &= \frac{1}{N} \sum_p \eta_p, \\
a &= \frac{1}{N} \left(\sum_p \zeta_p - b \sum_p \eta_p \right), \\
b &= \frac{\sum_p \{\zeta_p \left(\eta_p - \langle \eta \rangle\right)\}}{\sum_p \left(\eta_p - \langle \eta \rangle\right)^2}, \\
\sigma_b^2 &= \frac{\frac{1}{N-2} \sum_p (\zeta_p - a - b\eta_p)^2}{\sum_p \left(\eta_p - \langle \eta \rangle\right)^2}.
\end{aligned}$$

Although it may be interesting to check a as well, we only monitor b. The reason for this is the interpretation of deviations of b from unity. We are only interested in qualitative changes during optimization.

Eigenvalue Analysis - Correlation Among Hidden Units

We learned from our initialization procedure that the initial weights at the hidden units form mutually orthogonal vectors. This means that the outputs of the hidden units are uncorrelated. Since these outputs are linearly combined at the output node, uncorrelated outputs make "optimal" use of the hidden units. It is interesting to check whether the optimization procedure introduces correlations among the hidden units. If linear correlations exist this indicates redundant information. This redundancy is not desired at the input to the output layer, because it indicates a misspecification of the hidden layer. We therefore compute the *eigenvalues of the correlation matrix* of the hidden units outputs $v_l^{(p)}$. Let $y_l^{(p)}$ denote the activity of hidden unit l if pattern p is presented to the net, $y_l^{(p)} = a_{l0} + \sum_k a_{lk} \tilde{x}_{t-k}^{(p)}$. The output of this hidden node with this activity is denoted by $v_l^{(p)} = g(y_l^{(p)}) = \tanh\left(y_l^{(p)}\right)$. The correlation between node l and l' is computed as

$$\begin{aligned}
\tilde{v}_l^{(p)} &= \frac{v_l^{(p)} - \langle v_l \rangle}{\sigma_{v_l}}, \\
C_{ll'} &= \frac{1}{N} \sum_p \tilde{v}_l^{(p)} \tilde{v}_{l'}^{(p)},
\end{aligned}$$

where the first equation denotes standardization of outputs and the second equation denotes the element of the correlation matrix C. If there were any linear relations between these outputs, the

correlation matrix would show eigenvalues different from one. It may also be useful to interpret qualitative changes in the eigenvalues.

Singular Value Analysis - Identifying Linear Nodes

The Singular value decomposition (SVD) is the next technique we use to monitor the optimization process. It is applied to the weights at the hidden layer. SVD computes three matrices U, W, and V from the matrix A such that $A = UWV^T$. If A is a $M \times N$ matrix, then U is a column-orthogonal $M \times N$, W a $N \times N$ diagonal matrix with positive or zero elements, and V an orthogonal $N \times N$ matrix. The matrices U and V are each orthogonal in the sense that their columns are orthonormal. In compact form

$$A_{ij} = \sum_{k=1}^{N} w_k U_{ik} V_{jk}. \tag{2.130}$$

Here the columns of A are the weights at a specific unit, i.e. $A_{kl} = a_{lk}$ in notation of equation 2.121. Contrary to the correlation analysis, no pattern-based information is involved in this analysis. The diagonal elements of matrix W are of interest in our analysis. The singular values give qualitative information about the nonlinearity of the nodes and also about the specification of the hidden layer: Since we start from an initial solution with a known degree of nonlinearity, we may identify all singular values less than or equal to the initial singular values as belonging to essentially linear nodes. On the other hand, singular values (much) larger than the initial ones should belong to nonlinear nodes. We present an intuitive explanation for this reasoning. Since the matrices U and V are (column-)orthonormal, their elements can neither grow arbitrarily large nor extremely small, when the elements of A are modified by the optimization procedure. If the entries of A grow large (small), this can only be "compensated" by an increase (decrease) of diagonal elements of A. This argument also explains the qualitative nature of the interpretation of changes in the singular values. The important point here is that any model with more than one linear node is misspecified, because all linearity may be assigned to one single hidden unit.

Indications for Overfitting

If we introduce a validation set (e.g. by the clustering algorithm proposed above) then we can monitor its characteristics simultaneously. If training and validation set are initially similar, then discrepancies between the two sets developing during optimization indicate overfitting. This supports our proposition that the size of the training set needs to be chosen carefully.

Diagnostics

Diagnostic checking is an important step of statistical time series modelling. Usually it is ignored in the neural literature on time series analysis. Diagnostic checking can be formulated with respect to a single model and it can help to choose the best from a variety of models. Each of these is expected to show *iid* residuals with zero mean.

The problem of choosing the best specified model out of a set of individual models is addressed by *information criterion statistics*. If we were fitting polynomials to our data it would be a known observation that the more parameters we introduce the more accurate the fit will be.

A fit that is too perfect with polynomials would result in bad interpolation and extrapolation properties of the "model". This is serious in time series analysis, because the main use of these models lies in exactly these areas. The selection of models based on IC for this reason does not only include the goodness-of-fit. It also takes into account the number of parameters in the model as a penalty term. The basic formulation then is

$$IC = -\ \textit{maximum log likelihood} + \textit{number of free parameters in model}\ .$$

Among all models that with minimal IC will be selected. A rigorous mathematical derivation of these concepts, which relies on asymptotic properties of the estimators, can be performed for ARMA(p,q) processes. This analysis gives rise to slightly different formulations of the second IC term. We present two of the information criteria developed in this framework, the Akaike IC (AIC) and the Bayesian IC (BIC).

$$\begin{aligned} \mathrm{AIC(p,q)} &= N \ln \widehat{\sigma}^2_{p,q} + 2\,(p+q), && (2.131)\\ \mathrm{BIC(p,q)} &= N \ln \widehat{\sigma}^2_{p,q} + 2\,(p+q) \ln N, && (2.132) \end{aligned}$$

where $\widehat{\sigma}^2_{p,q}$ is the estimated residual variance and N is the number of observations entering the estimation, i.e. the number of time series points in our case. It is also customary to look at "normalized" information criteria, where the above equations are divided by the number of observations, e.g. NAIC $= \ln \widehat{\sigma}^2 + 2(p+q)/N$ [GaS81]. In applications of these criteria one is interested in absolute differences between AICs (BICs) of the order of one. Since the derivation of the information criterion relies on the linearity of the model, we use the criterion in a closely related sense. We denote this in the computed criterion by a prime (e.g. NAIC$'$).[4]

Antithetic Combination of Models

We will now discuss the question whether we can combine the information from different models to obtain an improvement in the modelling. We obtain different models for example from iterations of the modelling process as mentioned in Tong's paradigm. We will present two slightly different ways of combining the results of connectionist modelling. The first one does not take the existence of other models into account, the second one deals with information obtained from previous models. First of all, however, we will estimate the combination effect. Assume two models and their residuals. Pattern p has the desired value $x_{(p)}$, the estimated values are $\tilde{x}_{(p,i)}$, where $i = 1, 2$. We then have for the residuals $\tilde{\epsilon}_{(p,i)} = \tilde{x}_{(p,i)} - x_{(p)}$. The variances of the residuals read

$$\tilde{\sigma}^2_1 = \frac{1}{P}\sum_{p=1}^{P} \tilde{\epsilon}^2_{(p,1)} \qquad \tilde{\sigma}^2_2 = \frac{1}{P}\sum_{p=1}^{P} \tilde{\epsilon}^2_{(p,2)}. \tag{2.133}$$

Now consider the combined value $\tilde{x}_{(p,1/2)} = 1/2(\tilde{x}_{(p,1)} + \tilde{x}_{(p,2)})$. For its variance we compute

[4] The properties of a time series model can also be explored by a prediction on the predicted values (sometimes called "n step ahead forecasting"). If we were to reconstruct attractors we could check whether characteristic features like cyclic behavior are captured in the model.

$$\begin{aligned}\tilde{\sigma}^2_{1/2} &= \frac{1}{P}\sum_{p=1}^{P}\left(\frac{1}{2}\left(\tilde{x}_{(p,1)}+\tilde{x}_{(p,2)}\right)-x_{(p)}\right)^2 \\ &= \frac{1}{4}\tilde{\sigma}^2_1+\frac{1}{4}\tilde{\sigma}^2_2+\frac{1}{2P}\sum_{p=1}^{P}\tilde{\epsilon}_{(p,1)}\tilde{\epsilon}_{(p,2)}.\end{aligned}$$

If we compute expectation values from this equation we get

$$\langle\tilde{\epsilon}^2_{1/2}\rangle = \frac{1}{4}\langle\tilde{\epsilon}^2_{(1)}\rangle+\frac{1}{4}\langle\tilde{\epsilon}^2_{(2)}\rangle+\frac{1}{2}\langle\tilde{\epsilon}_{(1)}\tilde{\epsilon}_{(2)}\rangle. \tag{2.134}$$

If both residuals are uncorrelated and have the same expected variance, then the expected residual variance of the combined series is reduced by a factor of two. Furthermore, if they are anticorrelated the expected variance is even reduced to zero. This technique is called antithetic combination. deGroot and Würtz (1991) introduced this technique into connectionist time series analysis independently of Mani (1991). Now the strategy has to be a combination of models which are as uncorrelated as possible. The first attempt is to neglect this requirement. One then assumes that the various models are independent from each other. The other approach incorporates the knowledge taken from the first model by modifying the target values. If we require complete anticorrelation, $\tilde{\epsilon}_{(1)} = -\tilde{\epsilon}_{(2)}$, then the new target values are $x'_{(p)} = 2\,x_{(p)} - \tilde{x}_{(p,1)}$. This equation aims at anticorrelation between residuals of the two series. We would like to add the observation that a second model, for which $\tilde{\sigma}^2_1 \approx \tilde{\sigma}^2_2$ holds, may be difficult to estimate. In fact, if the first model were the "true" model, then, conceptually, the true values are computed by exactly this model and there is no space for a second model of equal quality (i.e. $\tilde{\sigma}^2_2 \gg \tilde{\sigma}^2_1$). A different approach could therefore try to model the residuals of the first model by setting $x'_{(p)} = x_{(p)} - \tilde{x}_{(p,1)}$ and then addition of the two models. Our antithetic construction, however, works well, because the true model is usually unknown in practical cases.

The CNAR Software Package

This package provides a complete implementation of the connectionist technique described in this section. The goal of this software development is to provide tools for the time series analyst which enable interactive data analysis and modelling. We point out some features of our software: The design principles for the software package were modularity and portability. Modularity is quite natural for time series analysis software, because the analysis itself proceeds in steps which are well separated from each other. Modularity also allows exclusion of certain steps of the analysis. For example the weight initialization part can be skipped. Then the software provides random initialization of the network parameters. Portability facilitates the software to be run in a server-client environment. While computationally intensive parts like searching for indicators or clustering can be performed on the more powerful machines in a network of computers, less intensive parts are executed on the "clients". The software runs on Unix based Sun workstations, Silicon Graphics machines and Convex and Cray supercomputers. The libraries of main programs and subroutines are written mainly in Fortran and partially in C. There are around 20,000 lines of code, some of which was obtained from public domain sources, most of which, however, was written by the authors themselves.

2.4.3 Case Study: Power Consumption Forecasts in West-Bohemia

The following case study deals with the prediction of electric power consumption in West-Bohemia in the Czech Republic and was done by Würtz, deGroot and Pelikan (1991). Accurate predictions of power consumption is an important task for the effective operation, planning and control of any electric energy systems. We are interested in short term forecasts especially in the load of the early morning period. The task is as follows: Given the consumption at 6:00 am predict the values for 8:00, 9:00, and 10:00 am, respectively. These forecasts allow to regulate and control the maximum peak level at the morning hours in a better economic way and already small improvements in the forecast performance of a quarter or half percent result in essential savings. Since an accurate prediction of power consumption is of relevant practical importance, there are many different forecasting models documented in literature, ranging from linear approaches, via knowledge-based systems, to artificial neural networks.

Data and Selection of Predictors

Here we use data recorded from June 1990 to December 1991. A record includes information on hourly loads and on averaged daily temperatures and cloud cover variables. The meteorological data were computed as a weighted sum from 22 stations located in the West-Bohemian area. Consumption is measured in MW, temperatures in ^{o}Celsius, and he cloud cover variables are discrete values between 0.0 and 1.0 in steps of 0.1, where the lowest value corresponds to a bright sky while the highest value 1.0 denotes the overcast sky.

The time series to be predicted $Y_t = \{Y_{t(i)}\}$ are the differences in consumption

$Y_{t(1)}$: load today 8:00 am − load today 6:00 am,
$Y_{t(2)}$: load today 9:00 am − load today 6:00 am,
$Y_{t(3)}$: load today 10:00 am − load today 6:00 am.

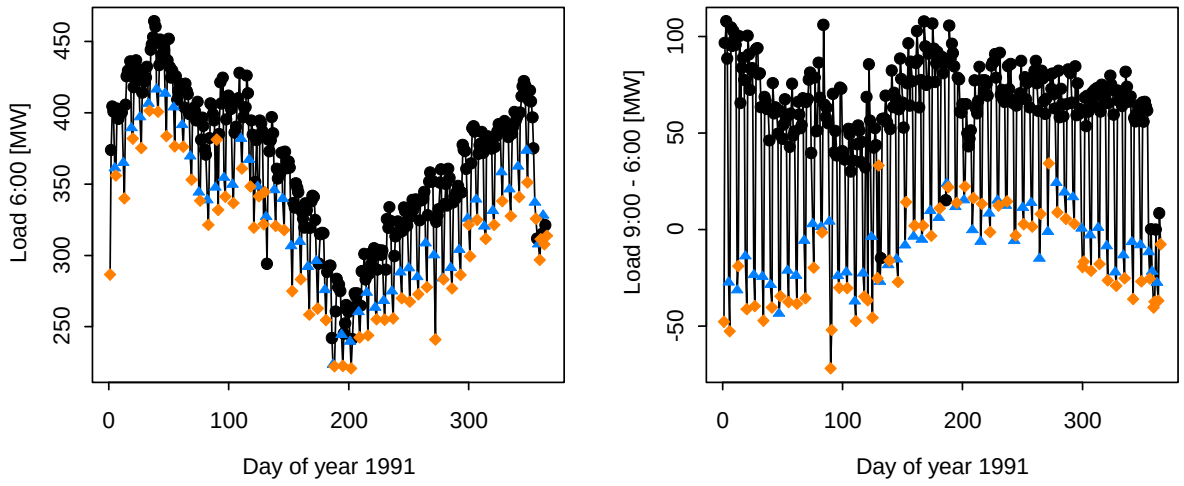
Exercise: Forecasting Power Consumption

Investigate the power consumption data, `power.dat` as supported by the `fSeries` Library. Calculate basic statistics, plot consumption and meteorological data as function of time.

The response function $Y_{t(2)}$ for 9:00 am is shown in figure 2.4.1 together with the load changes from 6:00 am to 9:00 am in figure 4.2.2. The 5:00 am load shows a quite regular behavior. The working days are well separated from the weekends. The highest power consumption is observed in the winter period during the first two weeks of February, the lowest during July which is summer vacation time. An irregular load can be identified during Christmas and the New Year period.

In order to identify predictors it is quite helpful to have a look onto cross-correlations of predictors and response variables. We consider the following set of predictors:

$X_{t(1)}$: load today 3:00 am
$X_{t(2)}$: load today 4:00 am
$X_{t(3)}$: load today 5:00 am
$X_{t(4)}$: load today 6:00 am
$X_{t(5)}$: load yesterday 8:00 am − 6:00 am
$X_{t(6)}$: load yesterday 9:00 am − 6:00 am
$X_{t(7)}$: load yesterday 10:00 am − 6:00 am
$X_{t(8)}$: load day before yesterday 8:00 am − 6:00 am

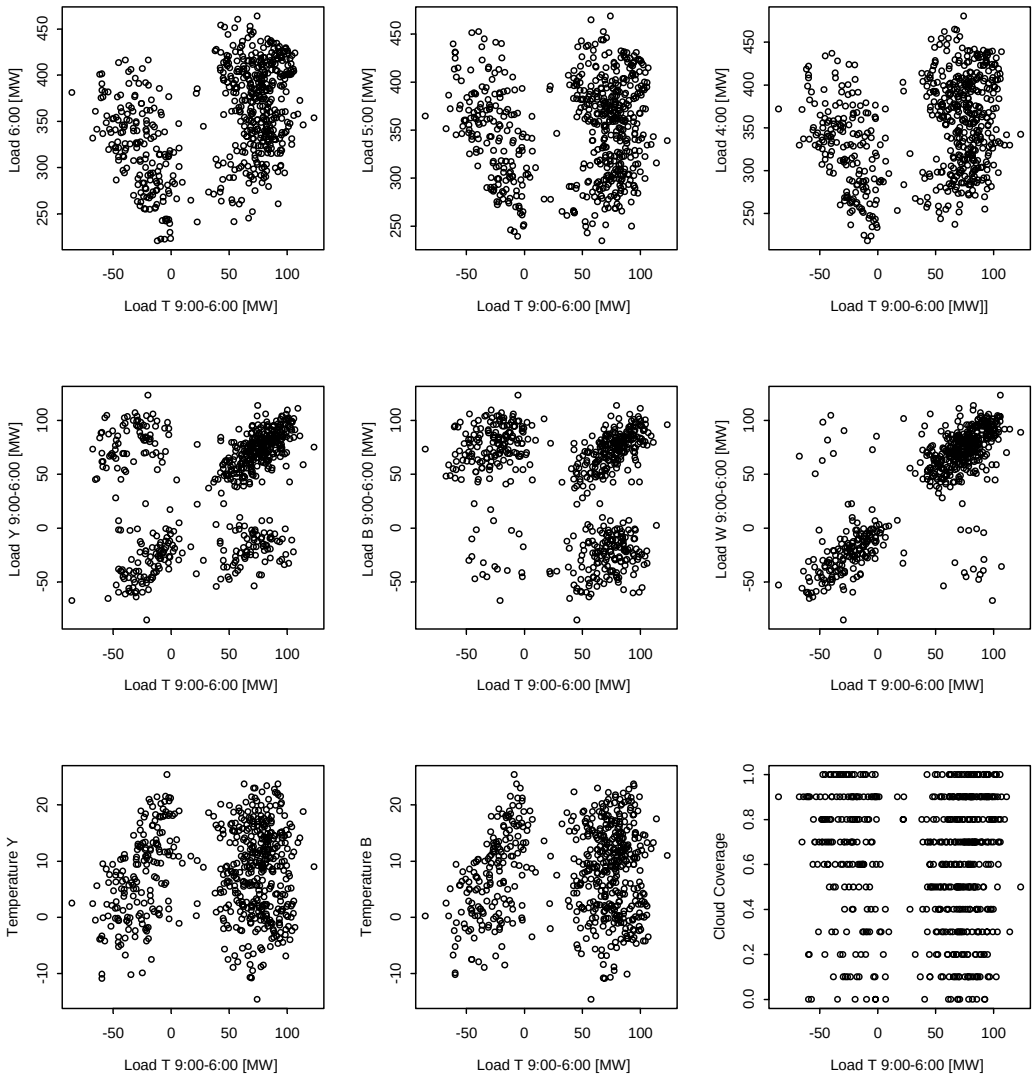


■ Figure 2.4.4: West-Bohemian power consumption data at 6:00 am in the year 1991. The full circles denote working days, Saturdays and Sundays are marked by a triangle and an octogon sign, respectively.
■ Figure 2.4.5: Changes in West-Bohemian power consumption between 6:00 am and 9:00 am in the year 1991. The plot symbols are the same as in the left figure.

$X_{t(9)}$: load day before yesterday 9:00 am − 6:00 am
$X_{t(10)}$: load day before yesterday 10:00 am − 6:00 am
$X_{t(11)}$: load a week ago 8:00 am − 6:00 am
$X_{t(12)}$: load a week ago 9:00 am − 6:00 am
$X_{t(13)}$: load a week ago 10:00 am − 6:00 am
$X_{t(14)}$: average temperature yesterday
$X_{t(15)}$: average temperature day before yesterday
$X_{t(16)}$: cloud cover yesterday
$X_{t(17)}$: cloud cover day before yesterday

Exercise: Forecasting Power Consumption, cont.

Build a set of predictors X and investigate their cross-correlation functions with the Y responses. Plot scatter diagrams.
Some of the cross-correlation scatter diagrams are shown in figure 2.4.6. These include consumptions at 4:00, 5:00, 6:00 am (first row); load changes between 6:00 and 9:00 am yesterday, day before yesterday, week ago (second row); temperature yesterday, day before yesterday, cloud coverage yesterday (third row). The scatter plots in the first row show two well separated clusters, one for the working and the other one for the weekend days. Within each cluster we see a clear correlation of the data. The next three scatter plots show the load differences. In the first plot of this sequence we detect four clusters; in the upper right corner we find the working days from Tuesday to Friday and in the other three corners of the plot we find Saturday, Sunday and Monday well separated. The second plot shows the working days Tuesday to Friday in the upper right corner, the weekend days in the upper left corner, and Monday and Tuesday in the lower right corner. In the "week ago" scatter plot we have the weekend on the diagonal in the lower part and the working days in the upper part. For the meteorological data in the last row we also observe strongly correlations. For the temperatures we observe again the separation of working and weekend days, for the cloud coverage data this observation is less pronounced.

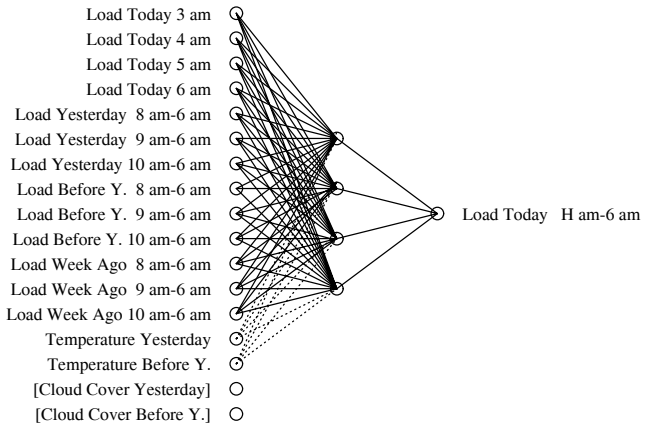


■ Figure 2.4.6: Scatter diagrams of selected predictors X vs. response $Y_{t(2)}$. The symbols are the same as in figure 2.4.4. The first row shows the consumption data at 6:00, 5:00, and 4:00 am. The second row displays the load changes between 6:00 and 9:00 am, one day, two days, and a week ago (**Y**: yesterday, **B**: day before yesterday, **W**: one week ago.) The last row illustrates meteorological dependencies, temperatures one ant two day ago, and yesterday's cloud coverage.

Neural Network Modelling

The setup for the neural networks as illustrated in Figure 2.4.3 includes as inputs consumption and temperature information, i.e. the standardized predictors $X_{t(1)} \dots X_{t(15}$ serve as inputs, and as response serves one of the three load differences $Y_t 1 \dots Y_{t(3)}$.

The data set consists of 579 data records. After excluding Christmas, New Year and national holidays, we are left with 324 days to build our models on. We decide to have a rolling time window for the modelling process. We select a number of days to estimate the model coefficients and then we use the model to predict the consumption for a certain number of days. Afterwards the window is shifted in such a way that the days we made a prediction on are included in the estimation, excluding the corresponding number of "oldest" data points. Then we estimate a



■ Figure 2.4.7: Setup of the CNAR network for modelling power consumption data. The first 13 indicators are used in a first step, then the next two are included. Cloud cover data dot not improve the forecast and therefore are left aside. The symbol **H** at the output denotes 8, 9, and 10 am, respectively. *Source: deGroot, (1993)*

new model, predict the consumption and move the window on, ready for the next model. We choose two sizes for the estimation window size, approximately three months (81 days) and approximately half a year (162 days). We use three modelling approaches, linear regression (LM) and a connectionist network with topology 13-4-1 or 15-4-1, where either the weights are randomly initialized (NN/RANDOM) or where the weights are initialized by the PCA scheme (NN/PCA). The results are shown in the following table:

Model:	Time:	Window Size: 81 Indicators		Window Size: 162 Indicators	
		1-13	1-15	1-13	1-15
Linear	8:00	2.80 %	2.73 %	2.59 %	2.49 %
Regression:	9:00	2.67 %	2.64 %	2.53 %	2.53 %
Model LM	10:00	2.94 %	2.87 %	2.67 %	2.59 %
Connectionist	8:00	3.85 %	3.44 %	2.90 %	2.92 %
Network:	9:00	2.96 %	4.19 %	2.92 %	2.80 %
Model NN/RAND	10:00	4.30 %	6.84 %	3.07 %	3.04 %
Connectionist	8:00	2.74 %	2.77 %	2.67 %	2.59 %
Network:	9:00	2.68 %	2.56 %	2.57 %	2.58 %
Model NN/PCA	10:00	2.96 %	2.79 %	2.72 %	2.67 %

■ Table: Average absolute errors of load forecasts in 1991. The first column describes the model, LM - linear modelling, NN/RAND - randomly initialized network, NN/PCA - initialized net by principal component analysis. Note, that in the case of the neural networks, the forecasts are the mean of 30 different network configurations. The second column gives the time, the third the results for a window size of 81 days and the last for a window size of 162 days. The left number in columns 3 and 4 exclude the temperature, while the right column includes this information. Note that a small but consistent improvement in the averaged error is achieved when we include the temperature information. A decrease of the error also occurs when we double the window size of the estimation from 81 to 162.

Exercise: Forecasting Power Consumption, cont.

> Can the generalized linear modelling, additive modelling, generalized additive modelling or the MARS modelling approach lead to a better forecasting performance? Reinvestigate the power consumption forecasts making use of these approaches.

Antithetic Combinations of Neural Networks

The simple neural network approach is not able to do essentially better forecasts than the linear model. Now we investigate if by antithetic combinations of neural networks the forecasts can be improved. To answer this question we concentrate on data around October 1991. The estimation period covers 162 days, and we include indicators 1–15. Now we vary the forecasting period to count 9, 18, 27, and 36 days. This means a single model is sufficient for daily forecasts on a period of 36 days and four models are required for nine days daily forecasts. After estimating one model on the original data, we change the target values in order to obtain optimally de-correlated patterns. We repeat this procedure for a third model.

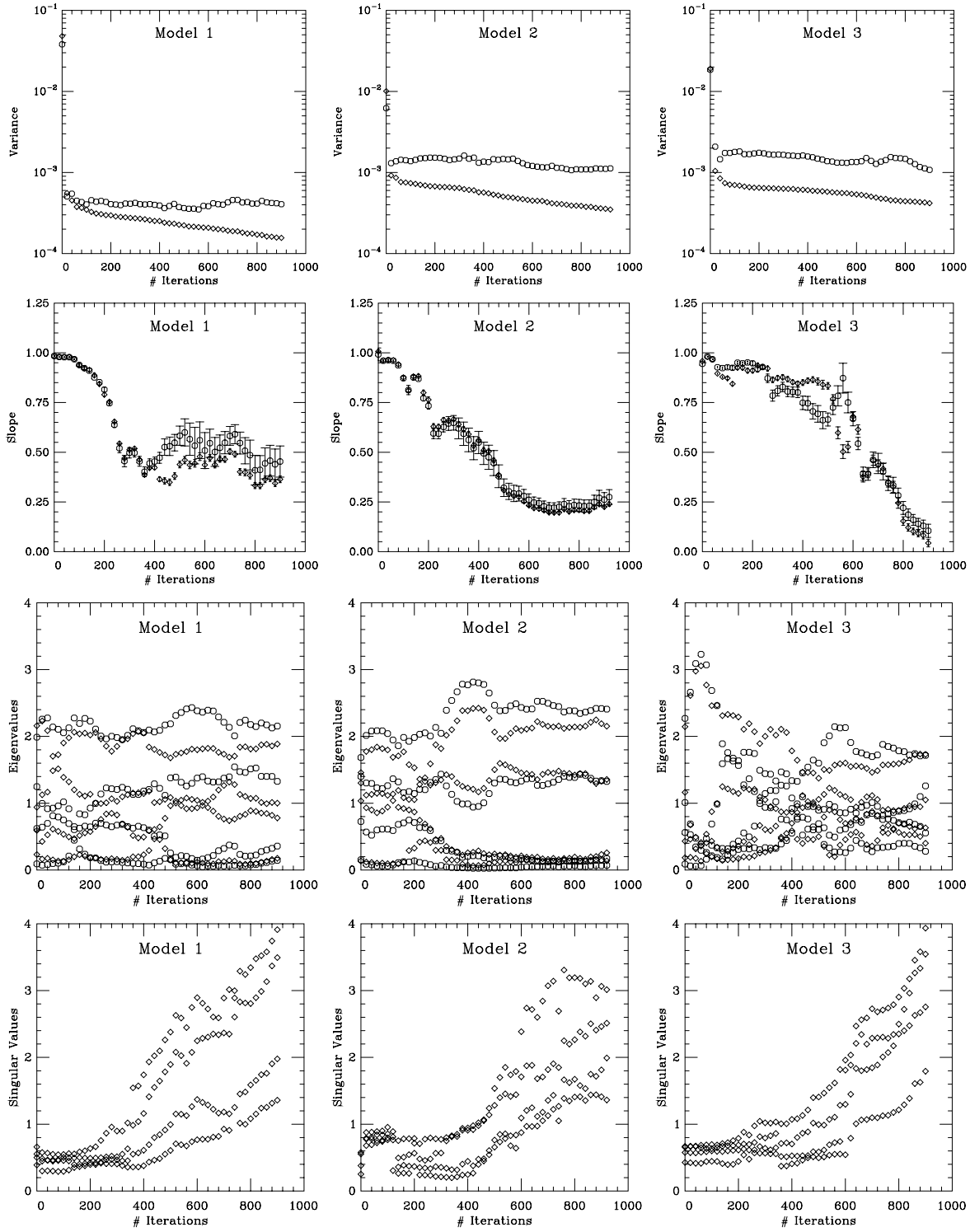
	Indicators 1-15, Window Size: 162			
Training				
162 days	1.43 % 0.488	1.60 % 0.140	1.56 % 0.481	1.13 %
Forecasting				
9 days	2.45 % 0.874	2.17 % 0.444	1.96 % 0.159	1.77 %
18 days	2.60 % 0.383	2.86 % 0.275	2.22 % -0.099	1.87 %
27 days	2.32 % 0.501	3.28 % 0.316	2.11 % -0.041	2.03 %
36 days	2.35 % 0.419	3.47 % 0.143	1.99 % 0.074	2.02 %
Note: Linear regression 9 days training error: 2.18 % Linear regression 9 days forecast error: 2.53 %				

■ Table: Absolute forecast errors for 9 am load and correlation coefficients of the residuals for the combination of three connectionist networks. The target values were modified in order to obtain maximally de-correlated models. The columns denote left to right errors for model 1, model 2, model 3, and the combination of all three. The second row in each line gives the correlation between model 1 and model 2, 2/3, and 3/1. For comparison the last row shows the error of the linear analysis.

The results of this analysis are given in table 2.4.2. In the first block from left to right we have the errors of models 1, 2, and 3 in percent, while the second line gives the correlation coefficients between model 1 and model 2, 2/3, and 1/3, respectively. The block on the right hand side contains the average error of a combination of all three models. For comparison the last row shows the error of the linear analysis. We observe very small correlations between the models. It is interesting to note that the de-correlation that was calculated on the training interval is also observed in the forecasting interval. In this instance the nonlinear models are better than the linear one. The improvement from an error of 2.53% down to an error of 1.77% is roughly 30%.

The NN Monitoring Scheme

Now we wish to explore our connectionist method for different values of the scaling parameter γ. We repeat the whole procedure with a smaller ($\gamma = 0.05$, model 2) and a larger value ($\gamma = 0.2$, model 3). We obtain average errors of about the same size. The differences between the models with different γ are mutually more similar than any of them with model R. — The existence of five different models gives rise to an investigation of the antithetic combinations. This analysis



■ Figure 2.4.8: Monitoring schemes for the optimization process as a function of the iterations for three decorrelated connectionist networks. From top to bottom we show: iteration path, slope of regression analysis, investigation of hidden correlation matrix, and singular value decomposition. A description can be found in the text. *Source: deGroot, (1993)*

is summarized in table 5.5. We find correlations coefficients between 0.83 and 0.95 if we compute them for all combinations of models L, 1, 2, and 3. If we combine these models, we observe a consistent improvement compared to the isolated models. As the correlation coefficient between model R and the other models ranges between 0.59 and 0.78, we also combine this model with the others, although model R shows quite large errors. Indeed we observe a consistent improvement. These observations explain also that the combination of all five models leads to the smallest errors.

For one instance of this last set of experiments we plot the monitoring in figure 5.20. The first row shows the *iteration path*, the second row the *slope of the regression analysis*, the third row the investigation of the *correlation matrix* and the last row the *singular value decomposition*. In the first row of the figures we observe a rapid decrease of the performance measure in the first few iteration steps of the optimization process for the training set (diamonds) as well for the forecasting set (octagons). This decrease is typical if the number of hidden units does not equal the number of input units. The sharp dropping is followed by a continuing decrease in the performance of the training set data albeit a less steep one, whereas the variance of the forecasting set remains almost constant. In the second row of the figures we plot the slope of the regression of the linear/nonlinear net versus iterations of the optimization algorithm together with the error estimate of the slope. The significant deviation of the slope from unity indicates that the three networks are operating in the nonlinear regime. The next row of the figures shows the time evolution of the eigenvalues of the correlation matrix during the optimization process. We observe for the training (diamonds) and forecasting set (octagons) a behavior that does not indicate linear correlations between the hidden units. This is least so in the plot for model 2. Here, we could try a restart with only three units. Overall, the eigenvalue analysis supports our choice of four hidden units. Let us finally look at the singular value decomposition of the weight matrix. After a few hundred iterations all singular values are larger than those of the initial network configuration. Note that this indicates an increase of nonlinearity which is independent of the particular pattern set. We also do not observe one or two dominating singular values. This is a further indication for the net utilizing all four hidden units.

In summarizing we note that the idea of antithetic combination of networks proves to yield quite good models in this case study. Prediction of load increase is possible with an error well below 2%. Our connectionist method results in much better models for the data than the models obtained from random initialization. Furthermore our method allows specification of four hidden units.

2.5 Design and Implementation of Trading System

Trade to establish the best track record, with steady gains and small drawdowns. -
The best trading systems are crude and robust. They are made of a few elements. The more complex systems, the more elements can break. -
A robust system holds up well when markets change.

Alexander Elder - Trading for a Living

Introduction

In this section we report on the design and implementation of an intra-daily trading system developed by D. Würtz and R. Schnidrig (1995, 1996). We describe how to calculate trading signals which give the entry points to trading positions and we describe in detail the strategy they used to decide how and when to exit a trade. The major elements of the trading system are also discussed: Using high frequency financial market data, working on an operational volatility based time scale and considering forecasts of the trading indicators. Furthermore, aspects of parameter selection, model evaluation and paper trading results are also presented.

To build a (real time) trading system includes a lot of different aspects: having reliable financial market data at hand, finding the proper timescale on which we enter or close trading positions, and calculating the recommendations combining technical trading rules with concepts from forecasting trading signals.

An important aspect is the timescale underlying the dynamics of the financial markets. Having the right "operational timescale" which holds the volatility of the time series essentially constant over time leads to a much more robust system compared to a trading model operating in physical time.

In a final step we introduce forecasts of indicators in designing trading systems. We include results from forecasts of trading indicators into the trading recommendations. This results in a much more reactive trading system. Traditional trading models consider only historical price movements, but don't do forecasts on indicators to learn about future price changes.

2.5.1 Trading Indicators, Trading Signals, and Trading Rules

The trading system takes quoted prices from a Reuters data selection feed. The foreign exchange markets trade mainly on the basis of two-way markets; i.e. market makers will simultaneously show prices, at which they will buy (bid) or sell (ask). In this scheme we distinguish four kinds of prices:

- *Bid Price:* The price at which a market maker will buy (their bid), P_t^{bid}.
- *Ask Price:* The price at which a market maker will sell (their offer), P_t^{ask}.
- *Middle Price:* The average price of the bid and ask price, $\overline{P}_t = (P_t^{bid} + P_t^{ask})/2$.
- *Current Price:* This price depends on the position of the trade. If the current position is neutral, we use the middle price, if long we use the bid price, and if short, we use the ask price.

Market practice dictates that the first price given, i.e. the one on the left, is the bid and the price on the right is the ask (offer). From quoted prices we calculate trading indicators from which we obtain trading signals according to given trading rules. These trading rules tell us which trading position we should take.

- *Trading Indicators* are time dependent functions usually calculated from historical prices. It is assumed that they are highly correlated to future price movements of the markets.
- *Trading Rules* are the result of strategies which combine different trading indicators to give a
- *Trading Signal* which is the recommendation to enter a new position, to reverse or close a position, or to change the current stop loss value. The
- *Trading Position* tells us if we are currently in a neutral (standing aside of the market), long (we have bought) or short (we have sold) position of a trade.

Our trading system combines a long term trend following indicator with a short term oscillator to follow the price movements. The price we use is the middle price $\overline{P}_t$ of the bid and ask. In principle any other monotonous increasing function, e.g. the logarithmic prices, $\log \overline{P}_t = 1/2(\log P_t^{bid} + \log P_t^{ask})$, can be used to calculate a trading indicator. In our case we use just the middle price.

We use a long and a short term indicator to filter out the disadvantages of both while preserving their strengths on the two different time scales. Our trading strategy demands that we examine the long term trend first. It allows us to trade only in the direction of the long term trend. It uses the short term oscillations for entering positions. Trend following indicators are used to identify long term trends. We use the *MACD* indicator to identify the direction of the market movement. The second step applies a short term oscillator to find deviations from the long term trend. The trading system allows us to take only those signals that point in the direction of the long term trend. We use a generalization of the *STOCHASTIC* indicator.

Entry Points

Entry points give the signals called "ENTER LONG" or "ENTER SHORT". They tell us when to enter from a "neutral" position to a new long (buy) or short (sell) position. Our entry points are determined from the interplay of a long term trend indicator, a short term oscillator and an overbought/oversold indicator.

Long Term Indicator

Trend following indicators are coincident or lagging indicators, i.e. they turn after trends have already reversed. We have selected the *MACD* indicator as our signal choice, since it can be calculated very easily from three *EMAs*. *EMAs*, Exponential Moving Averages, are trend following tools that give greater weight to the latest data and respond to changes faster than simple equally weighted moving averages. *EMAs* are calculated recursively in the following way:

$$EMA_\lambda(\overline{P}_t) = \lambda \overline{P}_t + (1-\lambda) EMA_\lambda(\overline{P}_{t-1}). \tag{2.135}$$

Here $\lambda = \frac{2}{(T+1)}$ denotes the decay length, where T is the length of the historical time window under consideration. These formulas tell us, that moving averages identify trends by filtering out daily price ripples.

MACD is an indicator build from three *EMAs* on different time horizons. The *MACD* indicator consists of two lines: the *MACD LINE* and the *SIGNAL LINE*. The *MACD LINE* is made of two *EMAs*. It responds to changes in prices relatively fast. The *SIGNAL LINE* is made of the *MACD LINE* smoothed with another *EMA*. It responds to changes in prices more slowly. The *MACD* indicator is constructed in the following way:

- Calculate a medium *EMA* (e.g. $T = 12$ days) of middle prices
- Calculate a long *EMA* (e.g. $T = 26$ days) of middle prices
- *MACD LINE*: Subtract the long from the medium *EMA*
- *SIGNAL LINE*: Calculate a short *EMA* (e.g. $T = 9$ days) of the *MACD LINE*

The numbers for the time horizons were taken as examples from Elder's book. *MACD* appears usually on price charts as two lines whose crossovers give trading signals. These crossovers between *MACD LINE* and *SIGNAL LINE* identify changing markets. Trading in the direction of a crossover means going in the direction of the market. Thus buy and sell signals are given when the *MACD LINE* crosses above or below the *SIGNAL LINE*. This behaviour can easily be checked by using the following trading indicator

Trading Indicator 1:

$$MACD(\overline{P}_t) := MACD\ LINE(\overline{P}_t) - SIGNAL\ LINE(\overline{P}_t)$$

Short Term Oscillator

STOCHASTIC is a short term oscillator very popular among traders. It tracks the relationship of the middle price to the recent high-low range. *STOCHASTIC* consists of two lines: a fast line called $\%K$ and a slow line called $\%D$. The first step is to calculate $\%K$[5]

$$\%K(\overline{P}_t) = \frac{2\overline{P}_t - (\overline{P}^{high}_{t-T\ldots t} + \overline{P}^{low}_{t-T\ldots t})}{\overline{P}^{high}_{t-T\ldots t} - \overline{P}^{low}_{t-T\ldots t}}, \tag{2.136}$$

where $\overline{P}_t$ is the middle price at time t and $\overline{P}^{high,low}_{t-T\ldots t}$ are the highest or lowest middle prices observed during the past time window of length T. The standard width for the *STOCHASTIC* indicator is $T = 5$ days.

The second step is to obtain $\%D(\overline{P}_t)$ which is done by smoothing $\%K(\overline{P}_t)$. There are several different possibilities how to do it, we prefer to use a simple recursive *EMA* with the standard time window of $T = 3$ days.

$$\%D(\overline{P}_t) = \lambda\%K(\overline{P}_t) + (1-\lambda)\%D(\overline{P}_{t-1}). \tag{2.137}$$

STOCHASTIC shows when the short term price movements become stronger or weaker.[6] This information will help us to decide whether an up or a down movement will dominate the current movement. Thus we evaluate as our trading indicator the expression

[5] A more computational effective way to calculate $\%K$ can be obtained from a generalization of this indicator: Evaluate the average of the high and low price from an *EMA* of prices and use for the the price range an *EMA* derived from the volatility.

[6] Other short term oscillators are for example Williams %R or the Relative Strength Index.

Trading Indicator 2:

$$STOCHASTIC(\overline{P}_t) = \%K(\overline{P}_t) - \%D(\overline{P}_t) \ .$$

Overbought/Oversold Indicator

In our case *STOCHASTIC* is designed to fluctuate between -1 and +1. We draw reference lines at -60% and +60% levels to mark "overbought" and "oversold" areas. An oscillator becomes overbought when it reaches a high level associated with tops in the past. Thus overbought means too high, ready to turn down. The vice versa holds in the case of oversold. We follow the suggestion of Alexander Elder: *Do not buy when STOCHASTIC is overbought and do not sell short when it is oversold. This rule filters out most bad trades.* We use $\%D$ as our overbought/oversold trading indicator.

Trading Indicator 3:

$$-0.60 < \%D(\overline{P}_t) < +0.60 \ .$$

Trading Rules:

The trading signals to enter a new position are determined from the above mentioned trading indicators. We enter a new long position, when one of the following two conditions is fulfilled.

Trading Rules 1 and 2:
ENTER_LONG(t):

```
IF                                IF
  POSITION(t-1) != +1               POSITION(t-1) != +1
  MACD(t) > 0                       MACD(t-1) < 0
  STOCHASTIC(t-1) < 0               MACD(t) > 0
  STOCHASTIC(t) > 0                 STOCHASTIC(t) > 0
  %D(t) < +0.6                      &D(t) < +0.6
THEN                              THEN
  POSITION(t) = +1                  POSITION(t) = +1
ENDIF                             ENDIF
```

In a similar way we enter a new short position, if one of the following two conditions holds.

Trading Rules 3 and 4:
ENTER_SHORT(t):

```
IF                                IF
  POSITION(t-1) != -1               POSITION(t-1) != -1
  MACD(t) < 0                       MACD(t-1) > 0
  STOCHASTIC(t-1) > 0               MACD(t) < 0
  STOCHASTIC(t) < 0                 TOCHASTIC(t) < 0
  %D(t) > -0.6                      %D(t) > -0.6
THEN                              THEN
  POSITION(t) = -1                  POSITION(t) = -1
ENDIF                             ENDIF
```

Exit Points

Exit points give signals called "GO NEUTRAL" or "STOP LONG" and "STOP SHORT". They tell us when it is time to exit a long (buy) or short (sell) position. We have two different kinds of exit signals:

- GO NEUTRAL when the long term trend reverses.
- STOP LONG or STOP SHORT when we get a signal from the stop loss order or from the profit take order.

Long Term Trend Reversion

The long term trend reverses when the *MACD* trading indicator changes sign. At this moment we "GO NEUTRAL" if we hold a long or a short trading position.

Trading Rule 5:

If we hold a long or a short trading position GO NEUTRAL when the MACD indicator changes its sign.

Stop Loss Order

We use a very tight stop loss strategy to minimize losses and to protect paper profits. It is worth to note, that a stop loss order limits the risk even though it does not always work. A stop is not a perfect tool but it is the best defensive tool a trader has. The rules for the initial stop loss at the entry point are the following:

- We place the stop price P_t^{stop} the time t when we enter the trade at price P_t^{enter}.
- We avoid all trades where a logical (percentual) stop S_t^{max} would expose more than 2%. The maximum S_t^{max} of 2% belongs to the bid or ask price depending on the position at which we enter the trade.
- We place our stop after entering a trade at the extreme $S_{t\ldots t-T}^{range}$ of the past $T = 5$ days' range but at maximum at S_t^{max}.

Elder suggests in his book *T=2* days. We take the same range as in the calculation of *%K*, just to simplify the calculations.

These stop loss orders can be summarized as follows:[7]

Stop Strategy 1: enter long position

$$\begin{aligned} P_t^{enter} &= P_t^{ask} \\ S_t^{range} &= \frac{\overline{P}_{t\ldots t-T}^{high} - \overline{P}_{t\ldots t-T}^{low}}{P_t^{enter}} \\ S_t^{max} &= 0.02 \\ S_t &= \min(S_t^{range}, S_t^{max}), \\ P_t^{stop} &= (1.0 - S_t)P_t^{enter} \end{aligned}$$

[7] Again, from a computational point of view, it is more efficient to replace in the stop strategies the range by an EMA of the volatility.

Stop Strategy 2: enter short position

$$\begin{aligned} P_t^{enter} &= P_t^{bid} \\ S_t^{range} &= \frac{\overline{P}_{t\ldots t-T}^{high} - \overline{P}_{t\ldots t-T}^{low}}{P_t^{enter}} \\ S_t^{max} &= 0.02 \\ S_t &= \min(S_t^{range}, S_t^{max}), \\ P_t^{stop} &= (1.0 + S_t) P_t^{enter} \end{aligned}$$

Here are some good reasons why 2% is a reasonable maximum loss value S_t^{max}:[8]

- Extensive testing has shown that the maximum amount a trader may lose on a single trade without damaging his or her long-term prospects is 2% of his or her equity.
- The 2% rule puts a solid floor under the amount of damage the market can do to your account. Even a string of five or or six losing trades will not cripple your prospects. In any case, if you are trading to create the best track record, you will not wan't to show more than 6% or 8% monthly loss.

Protect Profit Order

The protect profit order is part of an improved stop loss strategy. As soon as prices start to move in our favor we move in a first step the stop price in this direction.

Consider the time moving from t to $t + \tau$. If we are in a long position then $P_{t+\tau}^{max}$ denotes the maximum bid price on this time interval. If we are in a short position then $P_{t+\tau}^{min}$ denotes the minimum ask price on this time interval.

Stop Strategy 3: long position

$$\begin{aligned} P_{t+\tau}^{max} &= \max(P_t^{enter}, P_{t+1}^{bid}, \ldots, P_{t+\tau}^{bid}) \\ P_{t+\tau}^{stop} &= (1 - S_t) P_{t+\tau}^{max} \end{aligned}$$

Stop Strategy 4: short position

$$\begin{aligned} P_{t+\tau}^{min} &= \min(P_t^{enter}, P_{t+1}^{ask}, \ldots, P_{t+\tau}^{ask}) \\ P_{t+\tau}^{stop} &= (1 - S_t) P_{t+\tau}^{min} \end{aligned}$$

If we are moving beyond a "breakeven" point $S_t^{breakeven}$ (we use 0.5%, we start to protect efficiently paper profits.[9] Note, that paper profit in trading is real money, so one should treat it with the same care as realized returns and apply the following, so called 50%-rule: *Half the paper profit is ours and half belongs to the market. We mark the highest hight reached in a long trade or the lowest low reached in a short trade, and place our stop half-way between that point and our entry point.* The breakeven point and the 50%-rule are combined in the following way

[8] We made the same observation as described in the book of Alexander Elder. If we cut down the stop loss value below 2% we observe usually an increase in the number of trades resulting in a very likely decrease of the return.

[9] Elder suggests the following breakeven point: *As a rule, prices have to move away from your entry point by more than the average daily range before you move your stop to a breakeven level.*

Stop Strategy 5: long position

$$
\begin{aligned}
&S^{breakeven} = 0.005 \\
&S^{max}_{t\ldots t+\tau} = \frac{P^{\max}_{t\ldots t+\tau} - P^{enter}_t}{P^{enter}_t} \\
&P^{50\%-rule}_t = (P^{max}_{t\ldots t+\tau} + P^{enter}_t)/2. \\
&\\
&\text{IF } (S^{max}_{t\ldots t+\tau} > S^{breakeven} \text{ AND } P^{50\%-rule}_t > P^{stop}_{t\ldots t+\tau}) \\
&\qquad P^{stop}_{t\ldots t+\tau} = P^{50\%-rule}_t \\
&\\
&\text{ELSE} \\
&\qquad P^{stop}_{t\ldots t+\tau} = P^{stop}_{t\ldots t+\tau-1}
\end{aligned}
\tag{2.138}
$$

Stop Strategy 6: short position

$$
\begin{aligned}
&S^{breakeven} = 0.005 \\
&S^{min}_{t\ldots t+\tau} = \frac{P^{\max}_{t\ldots t+\tau} - P^{enter}_t}{P^{enter}_t} \\
&P^{50\%-rule}_t = (P^{min}_{t\ldots t+\tau} + P^{enter}_t)/2. \\
&\\
&\text{IF } (S^{min}_{t\ldots t+\tau} > S^{breakeven} \text{ AND } P^{50\%-rule}_t < P^{stop}_{t\ldots t+\tau}) \\
&\qquad P^{stop}_{t\ldots t+\tau} = P^{50\%-rule}_t \\
&\\
&\text{ELSE} \\
&\qquad P^{stop}_{t\ldots t+\tau} = P^{stop}_{t\ldots t+\tau-1}
\end{aligned}
\tag{2.139}
$$

The stop strategies can be summarized in form of the following

Trading Rule 6:

If we hold a long position and the current price (bid price) falls below the stop price P^{stop}_t then we close the position. If we hold a short position and the current price (ask price) goes above the stop price P^{stop}_t then we close the position.

Implementation of the System

There are three essential ideas underlying this trading system:

- Use high frequency financial market data.
- Work on an operational volatility based time scale.
- Forecast the trading indicators.

From high frequency data we explore the underlying dynamics of the price formation process from which we derive the operational time scale. Forecasting the trading indicators yields a more reactive system accompanied with lower drawdowns and shorter loss strings.

Using High Frequency Data

The use of high frequency data is a necessity in the design of intra-daily trading models. The environment for our realtime trime system is based on a Reuters data feed together with a highly sophisticated system for collecting archiving and validating hig frequency details. For details we refer D. Würtz and R. Scnidrig (1995) and for more details to the PhD Thesis of R. Schnidrig (1996).

Working on a Volatility Adjusted Time Scale

The operational υ-time scale is a volatility adjusted time scale on a weekly schedule. It conserves inbeetween of 168 weekly chosen datapoints on the average a constant volatility. The datapoints were derived from the scaling law. For the USDDEM exchange rate we calculated the following time schedule:

```
Monday:
    00:00 00:47 01:56 02:59 05:16 06:32 07:27 08:00 08:57 09:40 10:38 11:37 12:23 13:00
    13:35 14:03 14:23 14:40 15:00 15:22 15:45 16:12 16:44 17:13 17:35 18:08 18:55 19:43
    20:44 22:36
Tuesday:
    00:13 01:18 02:31 04:26 05:23 06:20 07:17 07:36 08:09 08:59 09:53 10:55 11:54 12:34
    13:01 13:26 13:48 14:08 14:21 14:33 14:47 15:07 15:36 16:04 16:21 16:38 17:13 17:55
    18:37 19:22 20:18 21:27 23:22
Wednesday:
    00:22 01:25 02:38 04:48 06:09 06:53 07:30 08:06 08:37 09:12 10:04 11:07 11:57 12:35
    13:06 13:34 13:57 14:14 14:27 14:38 14:51 15:08 15:25 15:47 16:17 16:52 17:28 18:07
    18:54 19:50 21:30 23:20
Thursday:
    00:51 01:55 02:55 05:16 06:21 07:17 07:48 08:34 09:16 09:42 10:23 11:14 12:03 12:23
    12:41 12:58 13:15 13:32 13:50 14:08 14:27 14:48 15:14 15:50 16:17 16:40 17:13 17:50
    18:27 19:05 20:02 20:58 22:33 23:56
Friday:
    00:41 02:00 04:07 05:50 06:34 07:16 08:00 08:34 09:06 09:43 10:38 11:35 12:05 12:21
    12:36 12:52 13:08 13:24 13:40 13:56 14:12 14:26 14:39 14:53 15:08 15:25 15:44 16:06
    16:33 17:01 17:31 18:03 18:49 19:55
Saturday/Sunday:
    03:40 23:59 20:23 21:56 23:29
```

■ Table: The table shows the weekly averaged operational "upsilon time" used as clock in the trading system. The time intervals are based on 60 minutes in υ time. The time intervals were derived from the scaling law of the volatility of the USDDEM currency relationship. All times are given in GMT.

Forecasts of Trading Signals

The third important aspect is to include results of forecasts of trading indicators into the trading recommendations. Traditional trading systems make use only of historical price data. For the indicators considered so far, we now assume that the dynamic process which underlies the price data may be a linear function over time for the next few timesteps:

$$\overline{P}_t = a_t + b_t t + \epsilon_t. \tag{2.140}$$

This requires an estimation $\widehat{a}(\overline{P}_t)$ and $\widehat{b}(\overline{P}_t)$ of the parameters a_t and b_t. A corrected estimation for the value $\widehat{a}(\overline{P}_t)$ can be calculated from

$$\widehat{a}(\overline{P}_t) = EMA_\lambda(\overline{P}_t) + \frac{1-\lambda}{\lambda}\widehat{b}(\overline{P}_t), \tag{2.141}$$

where b_t is the difference of two succeeding steps in claculating the *EMA* for the price:

$$b_t = EMA_\lambda(\overline{P}_t) - EMA_\lambda(\overline{P}_{t-1}). \tag{2.142}$$

This value will be evaluated from an exponential moving average

$$\widehat{b}(\overline{P}_t) := EMA_\lambda(b_t) = \lambda b_t + (1-\lambda)EMA_\lambda(b_{t-1}), \tag{2.143}$$

which will be used as an estimate for the slope b_t The forecasted price $\widehat{P}_{t+T}$ on a time horizon T is then simply given by:

$$\widehat{P}_{t+T} = \widehat{a}_t + \widehat{b}_t T. \tag{2.144}$$

The *MACD* long term trend indicator is forecasted in the following way

MACD Forecast Correction

Estimate $\widehat{a}_t^{medium}$ and $\widehat{b}_t^{medium}$ from a medium *EMA* (e.g. $T = 12$ days, same as in the MACD calculation). Iterate a new *EMA* for the forecasted prices $\widehat{P}_t^{medium}$.
Estimate $\widehat{a}_t^{long}$ and $\widehat{b}_t^{long}$ from a long *EMA* (e.g. $T = 26$ days, again the same as in the MACD calculation). Again, iterate a new *EMA* for the forecasted prices $\widehat{P}_t^{long}$.
MACD LINE forecast: Subtract the new medium from the new long *EMA* based on forecasted prices.
SIGNAL LINE forecast: Calculate a further new short *EMA* (e.g. with $T = 9$ days) of the forecast corrected *MACD LINE*.

For the *STOCHASTIC* short term oscillator we make use of the new prices from the forecasts of the *MACD* indicator. We take as the forecasted price simply the average value $\frac{1}{2}(\widehat{P}_t^{medium} + \widehat{P}_t^{long}) =: \widehat{P}_t$.

STOCHASTIC Forecast Correction

Calculate from the $\widehat{P}_t$ prices a forecast corrected $\widehat{\%K}$ indicator. The formula is the same as in the calculation of $\%K$ but using now forecasted high and low prices. The length of the range of $T = 5$ days is again the same as in the case of the $\%K$ calculation.
The forecast corrected $\widehat{\%D}$ value is evaluated from the forecast corrected $\widehat{\%K}$ indicator in the same way as $\%D$. The decay length of $T = 3$ days for the *EMA* is again the same as in the case of the $\%D$ calculation.

OVERBOUGHT/OVERSOLD Forecast Correction

Use the forecast corrected $\widehat{\%D}$ value.

The forecasts of the trading indicators are done at every time step on our v time scale. Then we apply the following rule to evaluate forecast corrected trading rules:

General Forecast Correction Rule

If the indicator and its forecast result in the same decision we use the indicator itself. If they are in contradicition we use the average value of the indicator and its forecast as the new indicator.

This seems to us a very simple strategy for implementing forecasts of indicators into a trading system. Further more sophisticated strategies can also be used and are under current investigation.

Parameter Selection

Parameter selection is one of the most important points in designing a trading system. Our aim is not to find an optimal set of parameters for a given currency relationship. Instead we like to find a set of identical parameters which leads to robust results for all currencies on all markets.

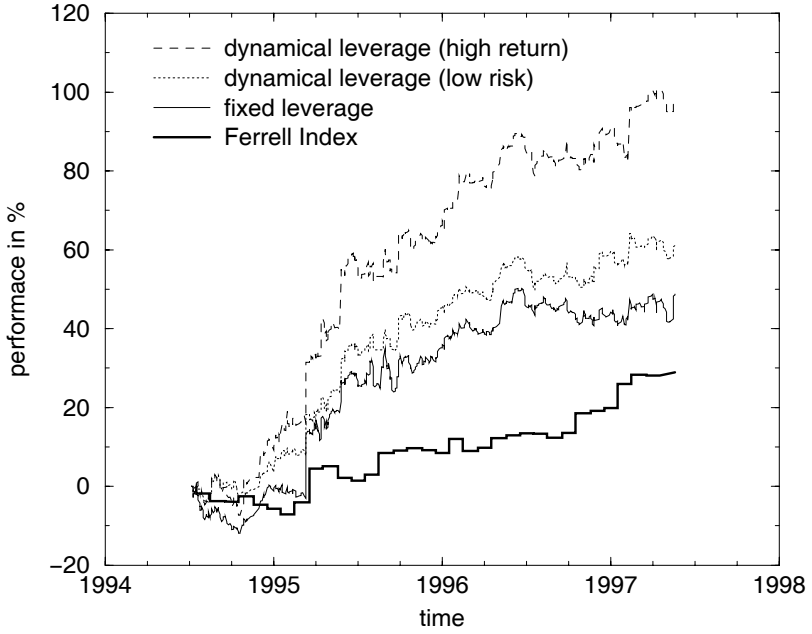
Therefore we have calculated only one "operational time scale" based on the most important market concerning the US Dollar and German Mark relationship. This timescale is applied to all our markets under consideration, even to the US/Japan market. We assume that this timescale is in a first approximation appropriate enough for taking major trading positions during european business times.

The *MACD* long term trend indicator should be also very robust. So we have selected as time constants the values of 26-12-9 days suggested by Alexander Elder. We have mapped these constants onto our operational v time scale. For the short term oscillator we use a 5 day period for the $\%K$ and a 3 day period for the $\%D$ *STOCHASTIC*. The time horizons are mapped again onto our operational v time scale. These numbers are very popular parameters which one can find in almost every trading textbook [1]. For the overbought and oversold signals we draw reference lines at -60% and 60% levels to mark the extreme areas. For the stop loss and profit take orders we use the following parameters: 2% for the maximum stop loss value, 0.5% for the breakeven point, and a 168 hours (5 days) window in v time to calculate the range used by the initial stop loss value. The time horizons in the case of forecasts have similar lengths as those used for the evaluation of the indicators based on historical data points.

Portfolio Currency Management

International currencies can be considered as an asset class with members having risk-return profiles that differ from each other and may be substantially uncorrelated. Like diversification within other asset classes, currencies may provide opportunities to diversify the foreign exchange portfolio of a currency manager.

Dynamical or tactical asset allocation is based upon the fact that expected revenues of the individual traders (models) and their volatilities among each other would change over time and therefore the optimal allocation should change. Thus each time when we get a new long or short signal from a trading model a new portfolio allocation is reevaluated to usually generate higher returns for the same level of risk.



■ Figure 2.6.1. The graph shows the performance of the trading system in comparison with the FX Ferrel Index (solid.line). Considered are the case of a fixed leverage and a dynamical leverage with preference to high return or to low risk, respectively.- *Source: D. Würtz et al.*

The usual approach to portfolio optimization is based on the mean variance portfolio approach introduced by Markowitz. The objective function considers the (negative) returns together with the risk evaluated from the covariance of the investments to be minimized. This has to be done under the conditions that individual investments are restricted to fall within a predetermined range and that their sum is limited by an upper bound. In addition a riskfree investment is easily added. Other portfolio optimization approaches allow for the optimization of the Sharpe ratio and may add penalty functions to lower drawdowns and to shorten loss strings. Further preferences of an investor can simply be added to the objective function within this approach. Here we present a first and very simple concept to improve risk-return profiles for currency investments. As an example we track exponential moving averages of the covariance matrix of the revenues of the individual traders. Whenever a new trading signal arrives, the objective function optimizes by a steepest descent approach (starting from an equally leveraged portfolio) the amount of individual currency investments. This is done by minimizing the deviation of the risk calculated from the covariance matrix in relation to a predefined risk value. As a further simplification at each optimization step no re-adjustment of the whole portfolio is performed, only the investment of the trader from which the arrived trading signal is adapted.

From the computational point of view this method is very attractive, since it allows an immediate availability of the optimized leverage factors after the new trading signal has arrived. Several hundred portfolios can be tracked in realtime by using this approach. This approach was applied to the four major currencies, USDDEM, USDJPY, USDCHF, and DEMJPY. Trading considers investments into the "FX-Triangle" America (USD) - Europe (DEM/2, CHF/2) - Asia (JPY) with 33% USDJPY 17% USDDEM, 17% USDCHF and 33% DEMJPY allocations. Since mid of 1996, the portfolio program has calculated, in realtime, dynamically leveraged portfolio recommendations which realize two different investor preferences: The first portfolio is designed to

achieve approximately the same return but with lower risk (average leverage factor 2.7 +/- 1.9, maximum leverage factor 10) as compared to the fixed leveraged portfolio (leverage factor 3), and the second portfolio tries to generate higher returns on the same risk level (average leverage factor 3.7 +/- 2.5). The results are shown in the Figure comparing the results of the dynamically leveraged portfolios with the fixed leveraged portfolio and the Ferrell Index benchmark. The results are quite impressive.

Nevertheless, money management rules can be applied on top of the portfolio approach to protect from large losses. These may be rules possibly formulated as: (1) never loose more than 5% per month, (2) never loose more than 10% per quarter, and (3) drawdowns should not exceed 15%. Furthermore, one can think about realizing paper profits at the end of a week or at the end of a month, to achieve a more continuous return profile over time which is usually more attractive to most of the investors.

2.6 The fSeries Library

2.6.1 Summary of Splus Functions

The following section gives an overview over the Splus functions available in the fSeries Library. The programs are grouped by their functionalities. A short description follows each Splus function name.

ARIMA Time Series Analysis (2.1)

Standard Splus offers several functions for the investigation of ARIMA times series. In the heart of these functions is the Gaussian Maximum Likelihood estimator for ARIMA process, whereas for the simple AR process also other estimators are available.

```
arima.sim           Simulates an univariate ARIMA time series
arima.mle           Returns a list by MLE representing an univariate ARIMA model
arima.diag          Computes diagnostics for an ARIMA model
arima.diag.plot     Plots diagnostics for an ARIMA model
arima.filt          Computes 1-step predictions and filtered values for an ARIMA model
arima.forecast      Forecasts an univariate time series using an ARIMA model

ar.yw               Fits an AR model using the Yule-Walker equations
ar.burg             Fits an AR model using the Burg's algorithm
ar.gm               Computes robust generalized M-estimates of AR parameters

acf                 Estimates and plots autocovariance, ACF and/or PACF
acf.plot            Plots autocovariance, ACF, PACF with inputs from acf or ar
```

The fSeries library adds the following functions:

```
trueacf             Estimates and plots the true ACF and/or PACF
trueacf.plot        Plots the true ACF and/or PACF
arma.roots          Computes and plots the roots of an AR or MA polynomial
```

GARCH Time Series Analysis (2.2)

Splus offers a GARCH module which is not part of the standard Splus package. The author has added two functions for simulation and estimation of APARCH processes. Functions for diagnosis analysis and forecasting are still missing.

```
garch.sim           Simulates an univariate APARCH time series
garch.mle           Returns a list by MLE representing an univariate APARCH model
```

Regression Modelling (2.3)

Standard Splus offers several functions for regression analysis with linear and additive models and their generalizations:

```
lm                  Returns an object that represents a fit of a linear model
lm.object           Represents the fit of a linear model
predict.lm          Predicts new examples by a linear model
summary.lm          Prints a summary report

glm                 Returns an object that represents a fit of generalized linear model
glm.object          Represents the fit of a generalized linear model
predict.glm         Predicts new examples by a linear model
summary.glm         Prints a summary report

gam                 Returns an object that is a fit of a generalized additive model
gam.object          Represents the fit of a generalized additive model
predict.gam         Predicts new examples by a linear model
summary.gam         Prints a summary report

ppr                 Returns an object that is a fit to a projection pursuit regression model
ppr.object          Represents the fit of a projection pursuit regression model
predict.ppr         Predicts new examples by a projection pursuit regression model
summary.ppr         Prints a summary report
```

Note, that generic functions such as `print` and `summary` have methods to show the results of the fit. Furthermore `ppr` was included from the R package `modreg`.

The R/Splus package `mda` contains MARS functions written by Hastie and Tibshirani (1995). These functions were coded from scratch, and did not use any of Friedman's original MARS code. The authors claim, that one obtains quite similar results to Friedman's program, but not exactly the same results. Friedman's ANOVA decomposition is not implemented nor are categorical predictors handled properly. The `mda` package is included in the `fSeries` distribution.

```
mars                Returns an object that represents a fit of a MARS model
predict.mars        Predicts new examples by a MARS model
summary.mars        Prints a summary report
```

There is another R/Splus package `polymars` available written by kooperberg and Stone to perform a stepwise regression using piecewise linear splines. Also this package is included in the `fSeries` distribution.

```
polymars            Returns an object that represents a fit of a POLYMARS model
predict.polymars    Predicts new examples by a PLYMARS model
summary.polymars    Prints a summary report
```

Neural Networks (2.4)

The R/Splus package `nnet` written by Ripley and added to the `fSeries` distribution provides functions for the analysis with a feedforward connectionist network. A quasi-Newton optimizer,

written in C, is used to train the net. The `nnet` package is also included in the `fSeries` distribution.

```
nnet                Fits a single-hidden-layer NN, possibly with skip-layer connections
predict.nnet        Predicts new examples by a trained neural net
summary.nnet        Prints a summary report
nnet.hess           Evaluates the Hessian of the specified neural network
```

Trading Indicators (2.5)

The fSeries Library provides the major trading indicators to perform a "technical analysis" on stocks or other financial instruments.

```
ema                 Calculates Exponential Moving Average
bias                Calculates EMA-bias
roc                 Calculates Rate of Change
osc                 Calculates EMA-Oscillator
mom                 Calculates Momentum
macd                Calculates MACD
cds                 Calculates MACD Signal Line
cdo                 Calculates MACD Oscillator
vohl                Calculates High/Low Volatility
vor                 Calculates Volatility Ratio
fpk                 Calculates fast percent K (\%K)
fpd                 Calculates fast percent D (\%D)
spd                 Calculates slow percent D (\%D)
apd                 Calculates averaged \%D
wpr                 Calculates Williams \%R
rsi                 Calculates Relative Strength Index
```

2.6.2 List of Splus Datasets

```
nyseres.csv         log returns of NYSE Composite index values
bmwres.csv          log returns of BMW stock prices
usrecession.csv     Recession data for the US economy
pacificstocks.csv   Stock Indexes for the Pacific Markets
```

2.6.3 List of Splus Examples

The material presented is illustrated by several Splus examples In the following we give a full list of all examples together with a short description:

```
xmpArimaSimulation          Simulating ARIMA time series processe
xmpArimaTrueAcf             Calculating the true ACF of ARMA time series
xmpArimaRoots               Computing the roots of AR and MA polynomials
xmpArimaIdentificationa     Identifying an ARMA time series process
xmpArimaEstimation          Estimating the parameters of an ARMA process
xmpArimaSelectiona          Selecting a proper ARMA time series model
xmpArimaDiagnostics         Performing a diagnostic check for an estimate ARMA model
xmpArimaForecasting         Forecasting the next steps in an ARMA time series process
xmpGarchInnovSim            Simulating an GARCH time series process
xmpAparchSim                Simulating an GARCH time series process
xmpGarchMLE                 Estimating the parameters of an APARCH process
xmpRegRecessionGLM          Fitting US Recession Data by the GLM approach
```

```
xmpRegRecessionGAM          Fitting US Recession Data by the GAM approach
xmpRegRecessionPPR          Fitting US Recession Data by the PPR approach
xmpRegRecessionPOLYMARS      Fitting US Recession Data by the POLYMARS approach
xmpRegRecessionGLM          Fitting US Recession Data by the NNET approach
```

2.6.4 Implemeted Software Packages

The `fSeries` was written by the author implementing beside his own functions additional functions and routines written by others and interfacing to other Splus packages.

tseries:

The functions from the R package `tseries` support time series analysis with emphasis on non-linear, non-stationary, and financial modelling. Most of the programs were written by A. Trapletti. We have implemented the `garch()` functions from this package into the `fSeries` library under the name `ngarch()` to circumvent double notations with the S+GARCH function having the same name.

The source is available from:
http://cran.r-project.org/src/contrib/PACKAGES.html

splusTS:

SPLUSTS is a collection of Splus functions for time series analysis provided by Meeker (1998). The goal of this package is to better integrate the Splus analytical and graphical tools and provide a simpler interface for doing Box-Jenkins type ARMA/ARIMA modelling and analysis. We borrowed the functions ... and have them integrated into the `fSeries` Library. Only a compiled version for MS Windows is abvailable,not the source code.

The program is available from:
http://www.public.iastate.edu/ stat451/splusts/splusts.html

mda:

The functions from the R package `mda` support flexible discriminant analysis (FDA), mixture discriminant analysis (MDA), multivariate additive regression splines (MARS), and additive spline models by adaptive backfitting (BRUTO). Authors of the software are Hastie and Tibshirani, for the R port Leisch, Hornik and Ripley.

The source is available from:
http://cran.r-project.org/src/contrib/PACKAGES.html

modreg:

The functions from the R package `modreg` support projection pursuit regression analysis. Author of the underlying Fortran software is Friedman, for the R port Ripley.

The source is available from:
http://cran.r-project.org/src/contrib/PACKAGES.html

polymars:

The functions from the S/R package `polymars` support polychotomous regression based on Multivariate Adaptive Regression Splines, (MARS). Authors are Kooperberg and O'Connor, for the R port Masarotto.

The source is available from:
http://cran.r-project.org/src/contrib/PACKAGES.html

nnet:

The functions from the R package `nnet` support projection pursuit regression analysis. Author of the software and for the R port is Ripley.

The source is available from:
http://cran.r-project.org/src/contrib/PACKAGES.html

2.6.5 Other Software Packages of Interest

mars 3.5

The Fortran MARS 3.5 software package written by Friedman which implements the original MARS algorithm is no longer available for a free download from the Internet. MARS is now offered as a commercial software product by Salford Systems. Salford has enhanced the original code with several new features and capabilities in collaboration with Friedman.

The source is available from:
www.salford-systems.com.

SNNS

SNNS, Stuttgart Neural Network Simulator, the is a software simulator for neural networks on PCs and Unix workstations developed at the University of Stuttgart. The goal of the SNNS project is to create an efficient and flexible simulation environment for research on and application of neural networks. The SNNS simulator consists of two main components: 1) simulator kernel written in C and 2) graphical user interface under the Window System X11. The simulator kernel operates on the internal network data structures of the neural nets and performs all operations of learning (optimization) and recall. It can also be used without the other parts as a C program embedded in custom applications. This makes its interesting to integrate the software under Splus or R. SNNS can be extended by the user with user defined activation functions, output functions, site functions and learning procedures, which are written as simple C programs and linked to the simulator kernel. Currently the following network architectures and learning procedures are included: *Backpropagation for feedforward networks, Counterpropagation, Quickprop, Backpercolation, RProp, Generalized radial basis functions (RBF), ART1, ART2,*

ARTMAP, Cascade Correlation, Recurrent Cascade Correlation, Dynamic LVQ, Backpropagation through time (for recurrent networks), Quickprop through time (for recurrent networks), Self-organizing maps (Kohonen maps), TDNN (time-delay networks) with Backpropagation, Jordan networks, Elman networks and extended hierarchical Elman networks, Associative Memory. The X11 graphical user interface XGUI, built on top of the kernel, gives a 2D and a 3D graphical representation of the neural networks and controls the kernel during the simulation run. In addition, the 2D user interface has an integrated network editor which can be used to directly create, manipulate and visualize neural nets in various ways.

The package is available from:
http://www-ra.informatik.uni-tuebingen.de/SNNS/.

Bibliography

[1] Baldi P. and Hornik H., *Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima*, Neural Networks 2, 53, 1989.

[2] Barclay Trading Group, *Barclay CTA - Currency Traders Index*, monthly in: Futures, News, Analysis and Strategies for Futures, Options and Derivatives Traders

[3] BCI, The Conference Board, *Business Cycle Indicators*, 14 pages, 1997.

[4] Beck, N., Jackman Smith, *Getting the Mean Right is a Good Thing: Generalized Additive Models*, Working Paper, 55 pages, 1997.

[5] Bera A.K., Higgins M.L., *ARCH Models: Properties, Estimation and Testing*, Journal of Economic Surveys 7, 305362, 1993.

[6] Bernt E.K., Hall B.H., Hall R.E., Hausman J.A., *Estimation and Inference in Nonlinear Structural Models*, Annals of Economic and Social Measurement 3-4, 653-665, 1974.

[7] Bollerslev T., *Generalized Autoregressive Conditional Heteroscedasticity*, Journal of Econometrics 31, 307327, 1986.

[8] Bollerslev T., Chou Y.C., Kroner K., *ARCH modeling in Finance: A Selective Review of the Theory and Empirical Evidence*, Journal of Econometrics 52, 201-224, 1992.

[9] Bollerslev T., Engle R.F., Nelson D., *ARCH Models*, in: Engle R.F., McFadden D., editors, Handbook of Econometrics, Vol. IV. Amsterdam, North-Holland, 1994.

[10] Box G.E.P., Jenkins G.M., *Time Series Analysis: Forecasting and Control*, Holden-Day, San Francisco, 1976.

[11] Breiman L., Friedman J.H., Olshen R.A., Stone C.J., *Classification and Regression Trees*, Chapman & Hall, New York, 1984.

[12] Camacho M., Perez-Quiros G., *This is what the Leading Indicators lead*, Working Paper, 25 pages, 2000.

[13] Casdagli M., *Nonlinear Prediction of Chaotic Time Series*, Physica D35, 335, 1989.

[14] Chauvet M., Potter S., *Forecasting Recessions Using the Yield Curve*, Working Paper, 33 pages, 2001.

[15] Cybenko G., *Approximations by Superpositions of Sigmoidal Functions*, Technical Report, No 856, University of Illinouis, 1988.

[16] deGroot C., Würtz D., *Analysis of Univariate Time Series wit Connectionist Nets: A case Study of Two Classical Examples*, Neurocomputing 3, 177, 1991.

[17] deGroot C., Würtz D., *Signal Processing and Noise Reduction with Connectionist Networks*, Helvetica Physica Acta 64, 948, 1991.

[18] deGroot C., Würtz D., *Forecasting Time Series with Connectionist Nets: Applications in Statistics, Signal Processing and Economics*, in: Lecture Notes in AI 604, edts. F. Belli and F.J. Radermacher, Springer, 1992.

[19] Dennis J.E, More J.J., *Quasi-Newton Methods*, Motivation and Theory. SIAM Rev. 19, 4689, 1977.

[20] Dennis J.E., Mei H.H.W., *Two New Unconstrained Optimization Algorithms which use Function and Gradient Values*, J. Optim. Theory Applic. 28, 453482, 1979.

[21] Dennis J.E., Gay D.M., Welsch R.E., *Algorithm 573 An Adaptive Nonlinear Least-Squares Algorithm*, ACM Transactions on Mathematical Software 7, 369383, 1981.

[22] Dwinnell W., *Exploring MARS: An Alternative to Neural Networks*, ?, 21-24, 2000.

[23] Eckmann J.P., Ruelle D., *Ergodic Theory of Chaos and Strange Attractors*, Revue of Modern Physics 57, 617, 1985.

[24] Elder A., *Trading for a Living*, Kogan Page Ltd., London, 1993.

[25] Embrechts P., Dacorogna M.M., Somorodnitzky G., M̈iller U.A., *How heavy are the tails of a stationary HARCH(k) process? A study of the moments*, Preprint, 31 pages, 1996.

[26] Engle R.F., *Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation*, Econometrica 50, 9871008, 1982.

[27] Engle R.F. (editor), *ARCH - Selected Readings*, Oxford University Press, 424 pages, 1995.

[28] Estrella A., and Mishkin F.S., *The Yield Curve as a Predictor of U.S. Recessions*, FRB New York, Current Issues in Economics and Finance, 6 pages, 1996.

[29] Faraway J.J., *Practical Regression and Anova using R*, Working Paper, 203 pages, 2000.

[30] FX Week, *The FX Ferrel Index*, from: Waters Information Services, Inc., published monthly in: FX Week, The Business of Foreign Exchange

[31] Filardo A.J., *How Reliable Are Recession Prediction Models?*, Economic Review, FRB of Kansas City, Second Quartar 1999.

[32] Fox J., *Logit and Probit Models*, from: Statistical Course, 71 pages, 2001.

[33] Friedman, J.H., Stuetzle W., *Projection Pursuit Regression*, Journal of the American Statistical Association, 76, 817-823, 1981.

[34] Friedman, J.H., *Multivariate adaptive regression splines (with discussion)*, Annals of Statistics, 19, 1-141, 1991.

[35] Friedman, J.H., *Tutorial: Getting Started with MART in Splus*, Working Paper, 24 pages, 2000.

[36] Funahashi K.I., *On the Approximate Realization of Continuous Mappings by Neural Networks*, Neural Networks 2, 183-192, 1989.

[37] Gey S., Nedelec E., *Model Selection for CART Regression*, Working Paper, 28 pages, 2000.

[38] Goldfarb D., *Factorized Variable Metric Methods for Unconstrained Optimization*, Math. Comput. 30, 796811, 1976.

[39] Harman G., Parameswaran P., Witt M., *Shares, Bonds, or Cash? Asset Allocation in the New Economy Using CART*, Working Paper, 28 pages, 2000.

[40] Hartigan J.A., Wong M.A., *A k-Means Clusdtering Algorithm*, Applied Statistics 28, 100, 1979.

[41] Hastie T., *Neural Networks*, Chapter I in: Encyclopedia of Bistatistics, Working Paper, 8 pages, 1996.

[42] Hegland M., McIntosh I.,Turlach B.A., *A Parallel Solver for Generalized Additive Models*, Working Paper, 19 pages, 1998.

[43] Hendry D.F., *Modelling UK Inflation 1875-1991*, Journal of Applied Econometrics 16, 255-275, 2001.

[44] Hornik K., Stinchcombe M. and White H., *Multilayer Forward Networks are Universal Function Approximators*, Neural Networks 2, 359, 1989.

[45] Hwang J.N, Lay S.R., Mächler M., Martin D., Schimert J., *Regression Modeling in Back-Propagation and Projection Pursuit Learning*, Working Paper, 28 pages, 1994.

[46] Johnson J., *The Projection Pursuit Algorithms and Neural Networks*, Working Paper, 9 pages, 1997.

[47] Johnston G., *SAS Software to Fit the Generalized Linear Model*, Working Paper, 8 pages, 1996.

[48] Klinke S., Grassmann J., *Projection Pursuit and Neural Networks*, Working Paper, 44 pages, 1998.

[49] Koenig E.F., *The New Recession: Will we see it coming?*, Working Paper, 10 pages, 1998.

[50] Koenig E.F., *New Economy, New Recession?*, Working Paper, 10 pages, 2002.

[51] Kooperberg C., Bose S., Stone, C.J. Polychotomous Regression Journal of the American Statistical Association, 92, 1997.

[52] Lapedes A., Farber R., *Nonlinear Signal Processing Using Neural Networks: Prediction and System Modelling*, Preprint TR LA-UR-87-2662, Los Alamos, 1987.

[53] Leung M.T., Daouk H., Chen A.S., *Forecasting stock indices: a comparison of classification and level estimation models*, International Journal of Forecasting, 16, 173-190, 1991.

[54] Lingjaerde O.C., Liestol K., *Generalized Projection Pursuit Regression*, SIAM Journal Sci. Computation 20, 844-857, 1998.

[55] Mani G., *Lowering Variance of Decisions by Using Artificial Neural Network Portfolios*, Neural Computation 3, 4841991.

[56] Malki H.A. and Moghaddamjoo A., *Using the Karhunen-Loeve Transformation in the Back-Propagation Training Algorithm*, IEEE Transaction Neural Networks 2, 162, 1991.

[57] Mandelbrot B., *The Variation of Certain Prices*, Journal of Business 36, 394-419, 1963.

[58] Meeker W.Q., *Graphical Tools for Exploring and Analyzing Data From ARIMA Time Series Models*, Working Paper, 73 pagesm 2001.

[59] McCullagh P., Nelder J.A, *Generalized Linear Models*, Chapman and Hall, London 1989.

[60] Müller U.A., Dacorogna M.M, Olsen R.B, Pictet O.V., Schwarz M. and Morgenegg C., *Statistical Study of Foreign Exchange Rates, Empirical Evidence of a Price Change Scaling Law, and Intraday Analysis,* Journal of Banking and Finance, 14, 1189-1208, 1990.

[61] Müller U.A., Dacorogna M.M, Davé R.D, Olsen R.B, Pictet O.V., von Weizsäcker J. E., *Volatilities of different time resolutions analyzing the dynamics of market components*, Journal of Empirical Finance, Preprint UAM.1995-01-12, 1995.

[62] Dacorogna M.M, Müller U.A., Olsen R.B, Pictet O.V., *Modelling Short-Term Volatility with GARCH and HARCH Models*, Preprint MMD.1997-01-08, 1997.

[63] Nielsen H.A., *LFLM - Splus/R Library for locally wighted fitting of linear models*, Working Paper, 21 pages, 1997.

[64] Oja E., *A simplified Neuron Model as a Principal Component Analyzer*, Journal of Mathematics and Bio;ogy 15, 267, 1982.

[65] Olsen & Associates (Editor), *High Frequency Data in Finance*, Proceedings of the "First International Conference on High Frequency Data in Finance", Volume 1-4, Zurich, 1995.

[66] Opsomer J.D., Kauermann D., *Weighted local polynomial regression, weighted additive models and local scoring*, Working Paper, 15 pages, 2000.

[67] Potts W.J.E., *Generalized Additive Neural Networks*, Working Paper, 7 pages, 1999.

[68] Priestley M.B., *Spectral Analysis and Time Series*, Academic Press, London, 1981.

[69] Proietti T., *Forecasting the US Unemployment Rate*, Working Paper, 354 pages, 2001.

[70] Qinlan J.R., *Bagging, Boosting, and C4.5*, Working Paper, 6 pages, 1996.

[71] Reuters, *Reuter Data Book* and *Money 2000*, Version 9.61, Reuters, 1992.

[72] Rodriguez G., *Linear Models for Continuous Data*, Course Material, Chapter 2, 64 pages, 2001.

[73] Rodriguez G., *Generalized Liner Model Theory*, Course Material, Appendix B, 14 pages, 2001.

[74] Rudebusch G.D, *Has a Recession already started?*, FRBSF Economic Letter, October Isssue, 3 pages, 2001.

[75] Rumelhart D.E., McClelland J.L., *Parallel distributed processing, exploration in the microstructure of cognbition*, Vols. 1&2, MIT Press, Cambridge, 1986.

[76] Rumelhart D.E., *Learning and Generalization*, Proceedings IEEE International Conference on Neural Networks, San Diego, 1988.

[77] Saito Y., Takeda Y., *Predicting the US Real GDP Growth Using Yield Spread of Corporate Bonds*, Bank of Japan, Working Paper, 25 pages, 2000.

[78] Schiff, A., *Why does the Yield Curve Predict Economic Activity?*, Working Paper, 12 pages, 1999.

[79] Schimek M.G., Turlach B.A., *Additive and Generalized Additive Models: a Survey*, Working Paper, 42 pages, 1997.

[80] R. Schnidrig and D. Würtz, *Investigation of the Volatility and Autocorrelation Function of the USDDEM Exchange Rate on Operational Time Scales*, IPS Report No. 95-04; in: Proceedings of the "First International Conference on High Frequency Data in Finance" Zurich, Ed. Olsen & Associates, March 29-31, 1995.

[81] Schröder M., *Einführung in die kurzfristige Zeitreihenprognose und Vergleich der einzelnen Verfahren*, in: P. Mertens (Hrsg.), Prognoserechnung, p. 7-39, Physica Verlag, Nürnberg, 1993.

[82] Schwert G.W., *Why does Stock Market Volatility Change over Time?*, Journal of Finance 44, 1115-54, 1989.

[83] Sephton P., *Forecasting Recessions: Can we do better on MARS?*, Federal Reserve Bank St. Louis, 11 pages, 2001.

[84] So Y., *A Tutorial on Logisdtic Regression*, SAS Institute, Working Paper, 6 pages, 2001.

[85] Stone C.J., Hansen H., Kooperberg C., Truong Y.K., *The use of polynomial splines and their tensor products in extended linear modeling*, Annals of Statistics, 1997.

[86] Stovanovic D., Vaughan M.D., *Yielding Clues about Recessions: The Yield Curve as a Forecasting Tool*, Federal Reserve Bank St. Louis, 2 pages, 2000.

[87] Subba Rao T., Gabr M.M., *An Introduction to Bi-Spectral Analysis and Bilinear Time Series Models*, Lecture Notes in Statistics Vol. 24, Springer Berlin, 1984.

[88] Sun J., *Projection Pursuit*, Working Paper, 11 pages, 2000.

[89] Taylor S.J., *Modeling Financial Time Series*, Wiley, New York, 1986.

[90] Therneau T.M., Atkinson E.J., *An Intoduction to Recursive Partitioning Using the RPART Routines*, Working Paper, 52 pages, 1997.

[91] Tong H., *Threshold Models in Non-Linear Time Series Analysis*, Lecture Notes in Statistics Vol. 21, Springer Berlin, 1983.

[92] Tong H., *Non-Linear Time Series*, University Press, Oxford, 1990.

[93] Weigend A., Rumelhart D.E., *Generalization Through Minimal Networks with Application to Forecasting*, Proceedings of Interface'91, Springer New York, 1990. 1990.

[94] Würtz D., Schnidrig R., Labermeier H., *Archiving of High Frequency Data from Global Financial Markets*, IPS Report No. 95-03; in: Proceedings of the "Priority Programme Informatics Research" Zurich 1994, Eds. K. Bauknecht and J.M. Grossenbacher, p. 68-75, SNF Bern 1994; and Neural Network World 5, 635, 1995

[95] Würtz D., Schnidrig R., Labermeier H., Hanf M. and Majmudar J., *Analyse und Vorhersage von Finanzmarktdaten*, IPS Report No. 95-26, 1995.

[96] Xiang D., *Fitting Generalized Additive Models with the GAM Procedure*, SAS Institute, Working Paper P256-26, 8 pages, 2001.

[97] Zeng L., *Prediction and Classification with Neural Network Models*, Working Paper, 25 pages, 1996.