Ann. Inst. Statist. Math.
Vol. 53, No. 1, 179–188 (2001)
electronic version from http://ise.wiwi.hu-berlin.de/∼rolf/webquant.pdf, layout differs from published version

WEB QUANTLETS FOR TIME SERIES ANALYSIS *

Wolfgang Härdle, Torsten Kleinow and Rolf Tschernig

Institut für Statistik und Ökonometrie, Wirtschaftswissenschaftliche Fakultät, Humboldt-Universität zu Berlin, Spandauer Strasse 1, D-10178 Berlin

(Received February 7, 2000; revised September 18, 2000)

Abstract. New and advanced methods for nonlinear time series analysis are in general not available in standard software packages. Their implementation requires substantial time, computing power as well as programming skills. In time series analysis such a scenario is given by a recently suggested nonparametric lag selection procedure for univariate nonlinear autoregressive models which is based on the Corrected Asymptotic Final Prediction Error. In this paper we suggest a worldwide Web based specific client/server architecture that provides empirical researchers with fast access to new methods and powerful computing environments without knowing the statistical computing language and the server location. This architecture is implemented using the XploRe Quantlet technology and illustrated for nonparametric lag selection. Access to the Quantlet computing service can be obtained via standard WWW browsers or a Java client. The XploRe Quantlet service can be helpful in constructing research books and interactive teaching environments as the electronic version of this paper, available from http://ise.wiwi.hu-berlin.de/∼rolf/webquant.pdf, demonstrates.

Key words and phrases: distributed computing, lag selection, nonparametric time series analysis, Quantlets, Web based computing.

1. Introduction

New methods in time series analysis involve often computationally intensive numerical operations and entail often highly complex mathematical arguments. Bootstrapping time series (e.g. Franke *et al.* (1998)) involves quite a bit of computing resources, for example. Nonparametric procedures such as wavelets (e.g. Härdle *et al.* (1998), Dahlhaus *et al.* (2000)) assume a certain degree of mathematical skills before they can be used for practical implementation. Both the computational and mathematical complexity slows down the process of real world applications for new time series methods. In this paper we propose an architecture that helps in the proliferation of new time series methods. More precisely, we suggest a design for computing that combines Java applets with the advantages of statistical programming languages. This is particularly useful for nonlinear time series analysis. These methods and the advances in computing technology have made it possible to avoid too restrictive assumptions for model construction. Consider, for example, the problem of modeling a univariate time series. For the last two decades it has been common for analyzing time series data to use the well developed toolbox of linear autoregressive moving-average (ARMA) and integrated autoregressive moving-average

*The authors acknowledge support by the Deutsche Forschungsgemeinschaft via Sonderforschungsbereich 373 "Quantifikation and Simulation ökonomischer Prozesse" at Humboldt-Universität zu Berlin.

(ARIMA) models, see e.g. Brockwell and Davis (1991). This model class, however, can only be adequate if one is exclusively interested in linear dependencies through time.

However, in many applications modeling asymmetric effects or regime-dependent dynamics is of central importance. Then linear models are an inadequate choice and one considers, for example in finance, conditional heteroskedastic nonlinear autoregressive processes instead. A general nonparametric time series model for an univariate stochastic process $\{Y_t\}_{t=1}^T$ is given by

$$Y_t = f(Y_{t-i_1}, Y_{t-i_2}, \dots, Y_{t-i_m}) + \sigma(Y_{t-i_1}, Y_{t-i_2}, \dots, Y_{t-i_m})\xi_t \tag{1.1}$$

where $\{\xi_t\}$ denotes an i.i.d. noise with zero mean and unit variance and the form of $f(\cdot)$, the conditional mean function, and $\sigma(\cdot)$, the conditional standard deviation, as well as the number of lags m and the lags itself $i_1, \dots, i_m$ are unknown.

Quite often, theory in the field of application does not indicate a reasonable choice of these functions and parameters and one has to apply statistical model selection methods which only recently have become available for general function classes (Vieu (1994), Yao and Tong (1994), Auestad and Tjøstheim (1990), Tjøstheim and Auestad (1994), Tschernig and Yang (2000)).

All of these procedures are rather complex and computer intensive especially for large data sets. Both factors substantially hinder the proliferation and application of these techniques. In the best case, the developer of a new method of this kind provides public access to the algorithm via his own home page or a method archive which is accessible via FTP, e.g. http://lib.stat.cmu.edu/S/, for programs written in S-Plus. However, in order to execute the algorithm, the user needs access to the software system that can execute or compile the program (e.g. S-Plus, GAUSS, FORTRAN). Moreover, their implementation may not always be an easy task nor may these programs always be well checked and documented.

These difficulties can be circumvented by a more Web based approach in which case a new method is implemented in Java. While this allows potential users to access an algorithm from almost any platform, this method suffers from three drawbacks. First, due to its generality, Java implemented programs may be slower than those written in more platform dependent languages. Thus, the user's computing facilities may be occupied more than necessary. Second, a developer of a method faces double programming since he has to write the code both in a proprietary software language and in Java. Third, if Java programs are accessed via any browser one does not have access to the resources of the local machine. For example, one cannot load data files from the local hard disk. Nakano (1998) provided only a partial solution to the latter problems since it still excludes the advantages of Java applets.

Here we suggest a new approach that combines the advantages of Java applets with the advantages of statistical programming languages in a distributed computing framework. Our ideas are illustrated with advanced statistical model selection methods for time series data.

In Section 2 we present a nonparametric identification method for nonlinear time series models. Section 3 presents our new Web based computing architecture. Section 4 provides an application of these ideas using the algorithm of Section 2.

2. Nonparametric identification of nonlinear time series models

Fitting a nonlinear time series model like (**??**) requires two steps:

(i) Lag selection: one has to choose the relevant lags $i_1, \dots, i_m$ including their number m for the autoregression function $f(\cdot)$ and the conditional standard deviation $\sigma(\cdot)$.
(ii) Function estimation: estimate $f(\cdot)$ and, if desired, $\sigma(\cdot)$ for the chosen lag vector.

Here we present a nonparametric lag and estimation procedure by Tschernig and Yang (2000), hereafter abbreviated as TY. For their asymptotic analysis they assume among other things that the stochastic process $\{Y_t\}_{t=1}^T$ is stationary and β-mixing. We first describe Step **??** of nonparametrically estimating $f(\cdot)$. Denote the vector that contains all selected lags by $\mathbf{X}_t = (Y_{t-i_1}, Y_{t-i_2}, \dots, Y_{t-i_m})^T$. The idea is to estimate a first order Taylor approximation of the unknown function $f(\cdot)$ around a given point $\mathbf{x}$. Since including observations $\mathbf{X}_t$ that are too far away from $\mathbf{x}$ would introduce a large bias, one weights the observations. Using the least squares principle the estimated function value $\widehat{f}(\mathbf{x}, h)$ is provided by the estimated constant $\widehat{c}_0$ of a local polynomial estimate around $\mathbf{x}$

$$\{\widehat{c}_0, \widehat{\mathbf{c}}\} = \arg\min_{\{c_0, \mathbf{c}\}} \sum_{t=i_m+1}^{T} \left\{Y_t - c_0 - (\mathbf{X}_t - \mathbf{x})^T \mathbf{c}\right\}^2 K_h(\mathbf{X}_t - \mathbf{x})$$

where K denotes the weighting function which is commonly called a kernel function and $K_h(\mathbf{X}_t - \mathbf{x}) = h^{-m} \prod_{i=1}^m K\{(X_{t,i} - x_i)/h\}$ is a product kernel. $\widehat{f}(\mathbf{x}, h) = \hat{c}_0$ is known as a local linear function estimator. Similarly one can estimate the conditional standard deviation $\sigma(\cdot)$, see Härdle and Tsybakov (1997) for details. The parameter h is called bandwidth parameter and controls the statistical bias-variance tradeoff. Its estimation will be explained below.

We now turn to the problem of selecting the relevant lags (Step **??**). For this step it is necessary to a priori specify a set of possible lag vectors by choosing the maximal lag M. Denote the full lag vector containing all lags up to M by $\mathbf{X}_{t,M} = (Y_{t-1}, Y_{t-2}, \dots, Y_{t-M})^T$. The lag selection task is now to eliminate from the full lag vector $\mathbf{X}_{t,M}$ all lags that are redundant. Note that the candidate lag vector $\mathbf{X}_t$ is a subvector of $\mathbf{X}_{t,M}$.

For comparing the quality of competing lag specifications, one needs an appropriate measure of fit, as for example the Final Prediction Error (FPE)

$$FPE(h, i_1, \dots, i_m) = E\left[\left(\breve{Y}_t - \widehat{f}(\breve{\mathbf{X}}_t, h)\right)^2 w(\breve{\mathbf{X}}_{t,M})\right]$$

where the process $\{\breve{Y}_t\}$ is independent of the process $\{Y_t\}$ but has the same stochastic properties and $w(\cdot)$ is an appropriate weight function. Let $||K||_2^2 = \int K(u)^2 du$ and $\sigma_K^2 = \int K(u)u^2 du$. TY (Theorem 2.1) derive the Asymptotic Final Prediction Error

$$AFPE(h) = A + \frac{||K||_2^{2m}}{(T - i_m)h^m} B + \frac{\sigma_K^4 h^4}{4} C \tag{2.1}$$

where the first, the second, and the third term on the right hand side denote the final prediction error for the true function, the expected variance and squared bias of the estimator, respectively.

The final prediction error for the true function A can be estimated by the sample average

$$\widehat{A}(h) = (T - i_m)^{-1} \sum_{t=i_m+1}^{T} \left\{y_t - \widehat{f}(\mathbf{X}_t, h)\right\}^2 w(\mathbf{X}_{t,M})$$

based on the local linear estimator $\widehat{f}(\mathbf{X}_t, h)$. The asymptotic properties of the lag selection method rely on the fact that the argument of $w(\cdot)$ is the full lag vector $\mathbf{X}_{t,M}$.

By balancing the variance-bias tradeoff one obtains the asymptotically optimal bandwidth h_{opt} which can be estimated by plug-in techniques (Yang and Tschernig (1999)). This requires to estimate the unknown constants B and C. A nonparametric estimate of B is obtained from

$$\widehat{B}(\widehat{h}_B) = (T - i_m)^{-1} \sum_{t=i_m+1}^{T} \left\{ Y_t - \widehat{f}(\mathbf{X}_t, \widehat{h}_B) \right\}^2 w(\mathbf{X}_{t,M}) / \widehat{\mu}(\mathbf{X}_t, \widehat{h}_B)$$

where $\widehat{\mu}(\cdot)$ is a Gaussian kernel estimator of the density $\mu(\cdot)$ and where $\widehat{h}_B$ is based on Silverman's (1986) rule-of-thumb bandwidth. Similar arguments apply to the estimation of C.

Inserting the plug-in bandwidth $\widehat{h}_{opt}$ and proper bias correction of $\widehat{A}(h)$ leads to the following estimator of (**??**)

$$AFPE = \widehat{A}(\widehat{h}_{opt}) + 2K(0)^m (T - i_m)^{-1} \widehat{h}_{opt}^{-m} \widehat{B}(\widehat{h}_B). \tag{2.2}$$

The second term in (**??**) is a penalty term that punishes overfitting or choosing superfluous lags. It decreases with sample size as h_{opt} is of order $T^{-1/(m+4)}$.

In order to select the adequate lag vector, one computes (**??**) for all possible lag combinations with $m \leq M$ and chooses the lag vector with the smallest $AFPE$. TY (Theorem 3.2) showed that this procedure is weakly consistent, i.e. the probability of choosing the correct lag vector if it is included in the set of lags considered approaches one with increasing sample size. This consistency result is based on the fact that the rate of the penalty term in (**??**) depends on the number of lags m. Thus, if one includes superfluous lags in addition to all correct ones, the rate of the penalty term becomes slower which implies that too large models are ruled out asymptotically.

Furthermore, TY show that asymptotically it is more likely to overfit than to underfit (miss some correct lags). In order to reduce overfitting and therefore increase correct fitting, they suggest to correct the $AFPE$ and estimate the Corrected Asymptotic FPE

$$CAFPE = AFPE \left\{ 1 + m(T - i_m)^{-4/(m+4)} \right\}. \tag{2.3}$$

The correction does not affect consistency while additional lags are punished more heavily in finite samples. One chooses the lag vector with the smallest $CAFPE$.

3. Web Quantlets

To facilitate access to and usage of new statistical methods such as for example $CAFPE$, we propose a Web based client/server architecture. This architecture combines advantages of Web based statistical computing which is becoming increasingly popular with the well known advantages of distributed computing and central data storage. Central data storage concepts are implemented in a variety of database servers.

Web based statistical computing has been fueled by collections of Java applets designed for usage in HTML pages. Examples are the VESTAC project at KU Leuven (Darius,P., Ottoy, J.-P., Solomin, A., Thas, O., Raeymaekers, B. and Michiels, S., A colection of applets for visualizing statistical concepts, KU Leuven, http://www.kuleuven.ac.be/ucs/java/)

or some applets at http://statlab.uni-heidelberg.de/projects/random/ Heidelberg. Mostly, these applets provide more or less interactive programs for particular tasks. While methods exclusively encoded in Java can be easily used on almost every computer platform, they must be written in Java leading frequently to doubled programming effort for a method provider. Moreover, for computationally intensive statistical procedures Java applets occupy substantial computational resources on the client side as programs written in Java are slower compared to platform-dependent code and all the computations are conducted on the client's machine.

One way to avoid the numerical disadvantages of applets is to shift the computation to a powerful server machine while leaving the graphical user interface on the client. This separation of computing and presentation is implemented by a client/server architecture. An example for a Web based computing environment, where the graphical user interface (GUI) is available via the WWW at http://www.math.montana.edu/Rweb/, is the Rweb project. Rweb provides the user with a CGI (Common Gateway Interface) based Web interface to the statistical analysis package R. This interface consists of an editor to enter R code for sophisticated users and a list of predefined functions and data sets which can be selected without the need of programming for the inexperienced users. The results are represented as static HTML pages containing text and images. Thus, one has access to remote computational resources, however, at the cost of limited graphical interaction. Furthermore this approach does not allow to directly access results on the server since the session is closed after sending the HTML code to the client. To avoid these drawbacks we replace the CGI based interface by a Java applet.

Our approach then shares with Java applets their easy access to methods but shifts the computational burden to a server by incorporating distributed computing. It provides *access* to (statistical) methods which are implemented on a server and data via the Internet. The methods which are called Quantlets can be accessed by standard HTML browsers (Netscape Navigator 4.5, MS Internet Explorer 4.0). The GUI is implemented in Java, thus ready to run on every Java enabled platform, such as Java enabled Web browsers, as well as on most popular operating systems.

In opposite to a HTML/CGI user interface Java gives the opportunity of interactive statistical analysis. The statistical plug-ins are implemented on the server side in C++ or a matrix oriented programming language, providing fast computation on the Web. From the user's point of view the Quantlet server behaves like a collection of Java classes but, due to server side computation, it is able to handle complex statistical problems. If a matrix oriented programming language is used, adding new Quantlets is easy.

3.1 *Providing methods via the Web*

One of the most important advantages of the proposed architecture is the easy and standardized way for publication of new methods. It contains three parts: i) an environment for the method provider, ii) access to computational resources to apply the new methods to data for the method user and iii) a user interface ensuring a comfortable usage of the developed methods. One therefore has relationships between the method provider, the Quantlet Server (QS) and the Quantlet Client (QC).

This architecture equips the method provider with a tool to develop and test his methods as well as to apply them to his own data. This tool enables him to use a familiar statistical programming language as well as the opportunity to use native code (`DLL` for Windows QS, `so` for Unix QS) written in any programming language. In principle the methods can be developed in any computing environment which is connectable to a QC.

In addition, our proposed architecture offers the possibility for the method developer to provide Web based access to his new methods where the design of the QC should allow to control the options of method users. This can be achieved if the QC is configurable to flexibly restrict access to the methods, e.g. by allowing the user only to change parameters and execute the method or by permitting him to edit the Quantlet and to execute its modification. In the first case the QC could behave like a Java applet designed for a particular task, while in the second case it can serve as a simple computing environment like a Web based interface to an existing software.

Another way for a method provider is to write his own client. This is useful for developers who want to provide a specialized environment for particular customers, e.g. an environment for analyzing financial data.

3.2 *Technical background*

To obtain the features introduced in the last subsection the proposed system consists of three parts. Besides the QS and the QC we suggest to add a middleware (QM) which resides on the server host and manages the QS–QC communication. The structure of this architecture is shown in Figure **??**.

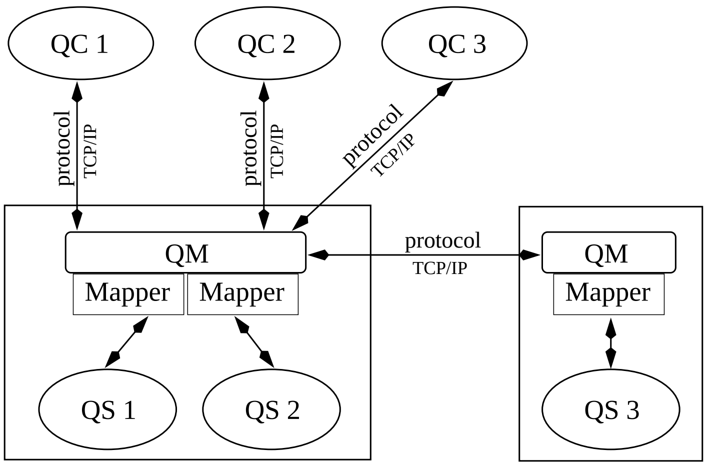


Fig. 1. The client/server (QC/QS) architecture

The main task of the QM is to handle data coming from the QS, transforming the data in a QC readable form and transmitting the data to the QC and vice versa. To have one middleware for multiple different QS's we have to ensure that the QM is available for a number of different platforms. Thus we implement it in Java. In addition Java contains packages for the TCP/IP communication and a class loader for dynamically loading classes at run time.

The data are transmitted from the QM to the QS via a TCP/IP socket. For this transmission we suggest a specific protocol. This protocol is based only on TCP/IP but not on any of its extensions or other derived protocols. This provides a system independent way of data transmission via the Internet. Both, the QM and the standardized protocol open the way to connect different clients with different servers by adding mappers to the QM. These mappers are Java classes which map the server specific output to the client readable protocol, i.e. they open the opportunity to connect the QC to a variety of different servers. This allows the use of every batch program as a server by transmitting data via its standard I/O streams. The class loader of Java provides the possibility to dynamically load these filters at runtime. To make the data transaction secure we suggest the use of the Java security package included in the Java Development Kit (JDK). Since the QM resides on the server host, all required operations for secure transactions can be done by the QM and do not have to be implemented on the mapper or the server. Besides the communication management the QM administrates system resources.

3.3 *The XploRe Quantlet Server*

The proposed QC/QS architecture has been implemented in XploRe (Härdle *et al.* (1999)). The QC, QS, QM and communication protocol introduced above are now given by i) an XploRe Quantlet Server (XQS) providing statistical computing power, ii) a Java based XploRe Quantlet Client (XQC) providing a user friendly graphical interface to interact with the XQS, iii) the middleware program MD*Serv and iv) the communication protocol called MD*Crypt.

As described, the XQS provides a statistical programming language (XploRe) and a variety of numerical Quantlets for statistical analysis. New Quantlets can be written either exclusively in the XploRe language or also as a combination of native code written in a programming language of one's choice via dynamic link libraries (`DLL`, `so`) and XploRe. The XQS writes the results of the computations into its standard Output stream. This stream is encoded in MD*Crypt, making the implementation of a mapper unnecessary. Via MD*Serv the client is able to decode MD*Crypt and to display the results in a readable form on the respective user platform. MD*Crypt is implemented in separate Java packages ready to be easily used in different clients written in Java. To open the possibility of native MS Windows clients or plug-ins to existing MS Windows software this package is also available as MS Windows DLL. At the moment we have one server and two clients which are available from http://www.xplore-stat.de. Besides the Java client there exists the ReX (XploRe, Excel) client based on Visual Basic for use of Quantlets in Excel. The ReX client uses the MS Windows DLL for communication.

4. Illustration

In this section we illustrate the suggested Web Quantlet architecture for the nonparametric model selection procedure which was presented in Section 2. First, we illustrate the statistical lag selection method via the Web. In this case, our architecture allows users or readers of the online version of this paper to replicate the results and modify chosen parameters of the executable Quantlets by clicking on the Quantlet symbol Q. In case of nonexecutable Quantlets the WWW based helpsystem is activated. Using the XploRe Quantlet Q genexpar we generated 50 observations of an exponential autoregressive process of order 2

$$Y_t = 0.3Y_{t-1} + 0.6Y_{t-2} + (1.9Y_{t-1} - 1.1Y_{t-2})\exp(-0.1 * Y_{t-1}^2) + \xi_t, \quad \xi_t \sim N(0,1)$$

which are shown in Figure **??**.

For this time series $CAFPE$ (**??**) is computed for *all* combinations of lags up to $M = 4$ as well as white noise and then the lag vector with the smallest $CAFPE$ is chosen. All this can be done by the Quantlet Qcafpe. It is implemented in XploRe and calls further XploRe Quantlets as well as a dynamic link libraries written in C++.

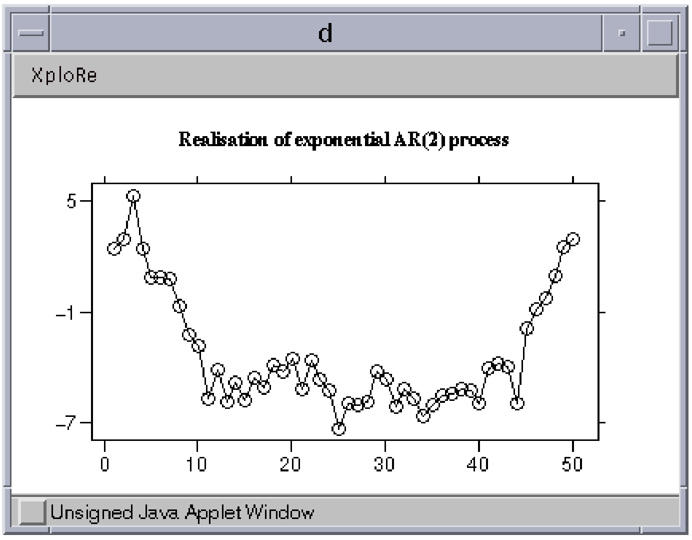


Fig. 2. 50 observations of the exponential AR(2) process

The size of the complete XploRe code is about 118k. Both Quantlets are called from the executable example program cafpeexample. This example program can be viewed, modified and executed from any Java capable Web browser such that a reader of the electronic version of this paper can obtain a similar output to Figures **??** and **??**. For the generated series, the method correctly selects lags 1 and 2 with $CAFPE = 1.4121$. Figure **??** also displays the best lag vector given the number of lags, i.e. for $m = 3$ lags 1,2,3 are selected with $CAFPE = 2.1042$.

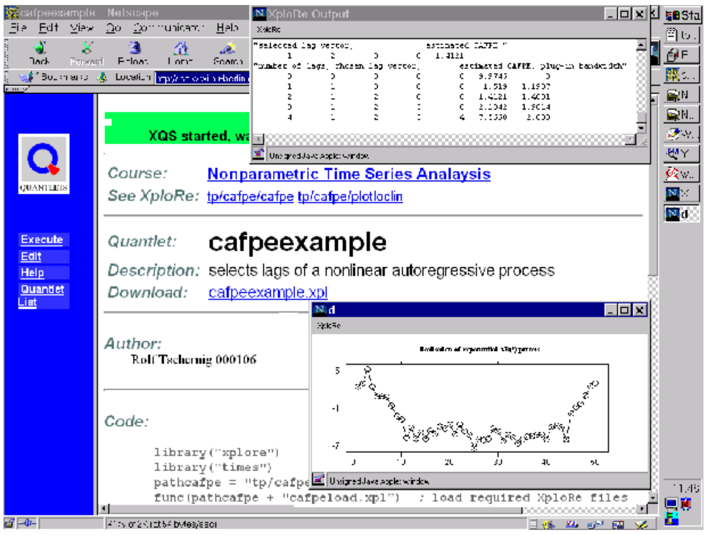
Once the lags are selected, one may estimate the regression function on a grid of the data by calling plotloclin from the executable example program plotloclinexample. This produces a 3-dimensional surface plot as shown in Figure **??** which can be rotated by the user. Here the rotation is done by the Java applet and computed on the client. If more than two lags are chosen, one can plot 3-dimensional surface plots if one conditions on the $m - 2$ least important lags.

If a reader is also interested in trying the $CAFPE$ Quantlet with his own data, he can download the XploRe client. This allows the user to access XploRe Quantlets as if it were installed on his local machine. Whatever approach is used, the client's machine is not occupied by the potentially computationally demanding Quantlets. Moreover, the server side computing provides the user with a powerful computing environment even if the client is somewhat outdated. The example with 5000 observations, for example, takes 10.5 hours on a Pentium 200 Mhz in contrast to 2.5 hours on the Sun Ultra 2 sparc server.

References

Auestad, B. and Tjøstheim, D. (1990). Identification of nonlinear time series: first order characterization and order determination, *Biometrika* **77**, 669–687.

Brockwell, P.J. and Davis, R.A. (1991). *Time Series: Theory and Methods*, Springer, New York.



Course: Nonparametric Time Series Analysis
See XploRe: tp/cafpe/cafpe tp/cafpe/plotloclin

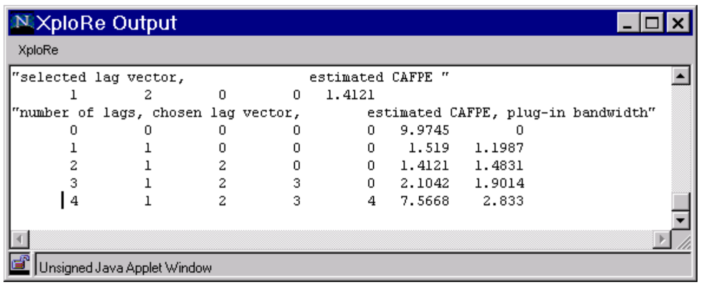
Quantlet: cafpeexample
Description: selects lags of a nonlinear autoregressive process
Download: cafpeexample.xpl

Author:
Rolf Tschernig 000106

Code:

```
library("xplore")
library("times")
pathcafpe = "tp/cafpe
func(pathcafpe + "cafpeload.xpl")   ; load required XploRe files
```

Fig. 3. Screen shot of running the example via the Web



```
"selected lag vector,                estimated CAFPE "
       1         2         0         0    1.4121
"number of lags, chosen lag vector,          estimated CAFPE, plug-in bandwidth"
       0         0         0         0         0    9.9745         0
       1         1         0         0         0     1.519    1.1987
       2         1         2         0         0    1.4121    1.4831
       3         1         2         3         0    2.1042    1.9014
       4         1         2         3         4    7.5668     2.833
```

Fig. 4. Results for the lag selection procedure

Dahlhaus, R., Neumann, M. H. and von Sachs, R. (1999). Nonlinear wavelet estimation of time-varying autoregressive processes, *Bernoulli*, 5, 873-906.

Franke, J., Kreiss, J.-P., Mammen, E. and Neumann, M. H. (1998). Properties of the nonparametric autoregressive bootstrap, Discussion Paper 54/98, SFB 373, Humboldt University, Berlin.

Härdle, W. and Tsybakov, A. (1997). Local polynomial estimators of the volatility function in nonparametric autoregression, *J. Econometrics* **81**, 223-242.

Härdle, W., Klinke, S. and Müller, M. (1999). *XploRe -The Statistical Computing Environment*, Springer, New York.

Härdle, W., Kerkyacharian, G., Picard, D. and Tsybakov, A. (1998). *Wavelets, Approximations, and Statistical Applications*, Springer, Heidelberg.

Nakano, J. (1998). Graphical user interface for statistical software using Internet, *COMPSTAT 1998 Proceedings in Computational Statistics* (eds. R. Payne and P. Green), 407-412.

Silverman, B. (1986). *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London.

Tjøstheim, D. and Auestad, B. (1994). Nonparametric identification of nonlinear time-series - selecting significant lags, *J. Amer. Statist. Assoc.*, **428**, 1410–1419.

Tschernig, R. and Yang, L. (2000). Nonparametric lag selection for time series, *J Time Ser. Anal.*, **21**,

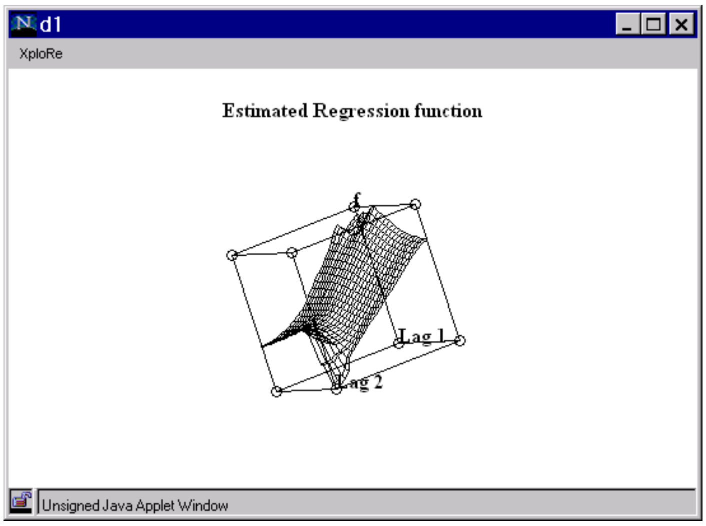


Fig. 5. Surface of the estimated two-dimensional regression function

457-487.

Vieu, P. (1994). Order choice in nonlinear autoregressive models, *Statistics*, **24**, 1–22.

Yang, L. and Tschernig, R. (1999). Multivariate bandwidth selection for local linear regression, *J. Roy. Statist. Soc., Ser. B*, **61**, 793–815.

Yao, Q. and Tong, H. (1994). On subset selection in non-parametric stochastic regression, *Statis. Sinica*, **4**, 51–70.