Detection of Changes in Financial Time Series

Rich Madsen
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/gradreports

Part of the Statistics and Probability Commons

Recommended Citation
https://digitalcommons.usu.edu/gradreports/1275

This Report is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
Detection of Changes in Financial Time Series

A Plan B Report submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Statistics

by

Rich Madsen

Approved:

Dr. Piotr Kokoszka
Major Professor

Dr. Richard Cutler
Committee Member

Dr. Russell Thompson
Committee Member

Utah State University, Department of Mathematics and Statistics
3900 Old Main Hill, Logan, UT 84322–3900
Spring Semester 2001
Detection of Changes in Financial Time Series

by

Rich Madsen

Utah State University, Department of Mathematics and Statistics
3900 Old Main Hill, Logan, UT 84322–3900
Spring Semester 2001
## Contents

1 Introduction .......................................................... 5
   1.1 Background on Time Series ........................................ 6
   1.2 Volatility ......................................................... 8
   1.3 The EWMA Model .................................................. 9
   1.4 The GARCH(1,1) Model ........................................... 10
       1.4.1 Mean Reversion ........................................... 10

2 Change Point Problem .................................................. 11
   2.1 Generalized Likelihood Ratio .................................... 11
       2.1.1 GLR Statistic for ARCH(1) Model ......................... 11
       2.1.2 GLR Statistic for GARCH(1,1) Model ..................... 15

3 Change-Point Tests Based on the Sequential Empirical Process .... 17
   3.1 Kolmogorov-Smirnov Type Statistic ............................ 17
   3.2 Cramer-von Mises Type Statistic ................................ 19

4 Analysis of Foreign Currency Data ................................... 20
   4.1 Returns on the DM Exchange Rate ............................... 20
   4.2 Returns on the British Pound Exchange Rate .................... 26

5 Checking the Power of the Change-Point Tests ....................... 28

6 Summary and Conclusions ............................................. 31

A Code used in formulating Table 1 output: .......................... 32

B Code used in formulating Table 2 output: .......................... 33

C Code used in formulating Table 4 output: .......................... 34

D Code used in formulating Tables 6 and 7 output: ......... 35

E Code used in formulating values for Figure 2 ......................... 36
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Code used in formulating values for Figure 3</td>
<td>37</td>
</tr>
<tr>
<td>G</td>
<td>C++ Code provided by Gudrun Kokoszka (GLRstat.h)</td>
<td>38</td>
</tr>
<tr>
<td>H</td>
<td>C++ Code provided by Gudrun Kokoszka (GLRstat.cc)</td>
<td>39</td>
</tr>
<tr>
<td>I</td>
<td>C++ Code provided by Gilles Teyssière (arch.h)</td>
<td>46</td>
</tr>
<tr>
<td>J</td>
<td>C++ Code provided by Gilles Teyssière (arch-main.ccp)</td>
<td>50</td>
</tr>
<tr>
<td>K</td>
<td>C++ Code provided by Gilles Teyssière (arch-dgp.cc)</td>
<td>62</td>
</tr>
<tr>
<td>L</td>
<td>C++ Code provided by Gilles Teyssière (arch-min.cc)</td>
<td>68</td>
</tr>
<tr>
<td>M</td>
<td>C++ Code provided by Gilles Teyssière (arch-hdnum.cc)</td>
<td>76</td>
</tr>
<tr>
<td>N</td>
<td>C++ Code provided by Gilles Teyssière (normal.cc)</td>
<td>85</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Time Series Plot of The U.S. Dollar vs. German Mark (Daily and Weekly Returns)</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>$\hat{M}_n$ Values vs. Time (Weeks) for the German Mark Data</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>$\hat{B}_n$ Values vs. Time (Weeks) for the German Mark Data</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Time Series Plot of The U.S. Dollar vs. British Pound (Daily and Weekly Returns)</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>Time Series Plot of Generated Data with Change Points at 300, 600 and 900</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>$\hat{M}_n$ and $\hat{B}_n$ Values From Generated Data with Change Points at 300, 600, and 900</td>
<td>30</td>
</tr>
</tbody>
</table>
List of Tables

1. Empirical biases and MSEs based on 1000 iterations ........................................ 6
2. $\Lambda_n$ Empirical quantiles for ARCH(1) model of size 100, based on 1000 iterations using the Splus code given in Appendix B ........................................ 13
3. $\Lambda_n^*$ Empirical quantiles for ARCH(1) model, based on 1000 iterations using the C++ program given in Appendix C ........................................ 14
4. Generated threshold $C$ values for different values of the level $\alpha$, based on 100 replications ........................................ 18
5. Picard’s threshold $C$ values for different values of the level $\alpha$ ................. 18
6. Checking Blum’s values, $F(y) = \lim_{n \to \infty} P_{H_0}\{\frac{1}{2} \pi^4 \hat{B}_n \leq y\}$, generated values based on 100 replications ........................................ 21
7. Checking critical values of Blum, generated values based on 100 replications .... 21
8. Segmentation Summary of German Mark Data Using $\hat{M}_n$ Statistic ............ 24
9. Segmentation Summary of German Mark Data Using $\hat{B}_n$ Statistic ............ 26
10. Segmentation Summary of Pound Data Using $\hat{M}_n$ Statistic ..................... 28
11. Segmentation Summary of British Pound Data Using $\hat{B}_n$ Statistic ............ 29
1 Introduction

The purpose of this paper is to examine and model data from several years of foreign currency trading, to determine if one or more change points has occurred in the data, and to estimate when those change points took place. Leading up to the analysis of the data we will construct and develop several statistics which we will use to determine if a change point has occurred.

This paper falls into the area of computational statistics and will make use of Splus and the S+GARCH module within Splus. Heavy use will also be made of C++. The models that we will be utilizing and discussing throughout the paper are the autoregressive conditional heteroscedasticity (ARCH) model and the generalized autoregressive conditional heteroscedasticity (GARCH) model. These specific models, along with several other similar models will be formally defined later in the paper.

With the GARCH module in Splus we are able not only to simulate ARCH and GARCH data, but also to estimate the parameters of these ARCH and GARCH models from the data. Naturally we are interested in the accuracy of these estimation techniques. If we are not able to accurately estimate the parameters of simulated ARCH and GARCH data, we can assume that we would likewise have difficulty in estimating parameters of data that is only proposed to follow these ARCH or GARCH models.

The first task is to assess the accuracy of the GARCH and ARCH modeling tools. Since we will be focusing our attention on ARCH(1) and GARCH(1,1) models, we focus on these models. We will generate several ARCH(1) and GARCH(1,1) data sets, estimate the parameters of the generated data sets, and compare those estimates with the actual parameters. Although we have not yet formally defined these models, this will give us an indication of the possible bias and the mean-squared-errors of our modeling tools for different sample data sets. We use generated data sets of size 100, 250 and 500 to roughly approximate data from a partial year, a full year and two full years of daily returns. 1000 models and corresponding estimates for each of the parameters were computed in order to produce the empirical biases and the MSE’s of the estimates as shown in Table 1. The Splus code has also been supplied for the reader in Appendix A. It is easily seen that the bias and MSE for our parameters are quite small indicating that with generated ARCH(1) and GARCH(1,1) data, we are fairly accurate at estimating the parameters. Particularly for larger data sets, it appears that our estimates are quite similar to the actual parameters. This is important in that we can now continue with our analysis while having the confidence and knowledge that our estimates are fairly accurate. This is an important conclusion and we will use this assumption of accurate estimates later on in the paper.

We note that, as defined later in equations (10) and (19), an ARCH(1) model has only two parameters namely $\omega$ and $\alpha$, and a GARCH(1,1) model has three: $\omega$, $\alpha$, and $\beta$.

Before we continue, some background and explanation of time series theory is helpful at this point.
Table 1: Empirical biases and MSEs based on 1000 iterations

<table>
<thead>
<tr>
<th>Model 1</th>
<th>$\omega$ Bias:</th>
<th>$n=100$</th>
<th>$n=250$</th>
<th>$n=500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCH(1)</td>
<td>$\alpha$ Bias:</td>
<td>-0.00625919</td>
<td>-0.00300568</td>
<td>0.00084227</td>
</tr>
<tr>
<td></td>
<td>$\beta$ Bias:</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>$\omega=0.2$</td>
<td>$\omega$ MSE:</td>
<td>0.00230028</td>
<td>0.00084289</td>
<td>0.00037563</td>
</tr>
<tr>
<td>$\alpha=0.5$</td>
<td>$\alpha$ MSE:</td>
<td>0.04220496</td>
<td>0.01564091</td>
<td>0.00843469</td>
</tr>
<tr>
<td>$\beta=0.0$</td>
<td>$\beta$ MSE:</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model 2</th>
<th>$\omega$ Bias:</th>
<th>$n=100$</th>
<th>$n=250$</th>
<th>$n=500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GARCH(1,1)</td>
<td>$\alpha$ Bias:</td>
<td>-0.01377061</td>
<td>-0.00471152</td>
<td>-0.00183119</td>
</tr>
<tr>
<td></td>
<td>$\beta$ Bias:</td>
<td>0.17172030</td>
<td>0.06434108</td>
<td>0.02377199</td>
</tr>
<tr>
<td>$\omega=0.01$</td>
<td>$\omega$ MSE:</td>
<td>0.00102213</td>
<td>0.00017538</td>
<td>0.00003385</td>
</tr>
<tr>
<td>$\alpha=0.2$</td>
<td>$\alpha$ MSE:</td>
<td>0.02119715</td>
<td>0.00629670</td>
<td>0.00300589</td>
</tr>
<tr>
<td>$\beta=0.7$</td>
<td>$\beta$ MSE:</td>
<td>0.17207210</td>
<td>0.03761940</td>
<td>0.00917078</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model 3</th>
<th>$\omega$ Bias:</th>
<th>$n=100$</th>
<th>$n=250$</th>
<th>$n=500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GARCH(1,1)</td>
<td>$\alpha$ Bias:</td>
<td>-0.05241303</td>
<td>-0.01794875</td>
<td>-0.00183119</td>
</tr>
<tr>
<td></td>
<td>$\beta$ Bias:</td>
<td>0.02553195</td>
<td>0.00958077</td>
<td>0.00357226</td>
</tr>
<tr>
<td>$\omega=0.3$</td>
<td>$\omega$ MSE:</td>
<td>0.04030998</td>
<td>0.01120102</td>
<td>0.00595363</td>
</tr>
<tr>
<td>$\alpha=0.7$</td>
<td>$\alpha$ MSE:</td>
<td>0.04129385</td>
<td>0.00871460</td>
<td>0.00374612</td>
</tr>
<tr>
<td>$\beta=0.2$</td>
<td>$\beta$ MSE:</td>
<td>0.05930643</td>
<td>0.02177034</td>
<td>0.01054067</td>
</tr>
</tbody>
</table>

1.1 Background on Time Series

Chatfield [3] defines a time series as “collection of observations made sequentially in time”. The closing price of a foreign currency, taken on successive days, fits this definition and is also defined as a discrete time series since the observations are taken only at specific times. A continuous time series, on the other hand, is one where observations are made continuously in time. A unique property of time-series is that the contiguous observations are dependent. Due to this dependency, we are able to predict the future values from past observations. Time-series of this type are considered stochastic. If we were able to predict exactly the future values of a time-series, it would be considered deterministic. Unfortunately we are not able to predict with exactness the future closing price of, say, a foreign currency.

Other properties of interest concerning time-series analysis include stationarity and non-stationarity. A time-series is considered stationary if there is no systematic change in mean, if there
is no systematic change in variance, and if the strictly periodic variations have been removed.

There are several classical probability models for time-series, some are presented in this section while others are discussed later in order to introduce certain ideas. These probability models are called stochastic processes and we begin with the autoregressive process. An autoregressive process of order \( k \), abbreviated AR(\( k \)), is denoted by:

\[
X_t = \phi_1 X_{t-1} + \ldots + \phi_k X_{t-k} + \epsilon_t, \quad \text{where } \epsilon_t \sim N(0, \sigma^2).
\]

From this we see that \( X_t \) is regressed on past values of \( X_t \). Thus this process should be applied when it is believed that present values are dependent on immediate past values along with a random error. We note that the simple first order case of AR(1) is often called the Markov process and is defined as

\[
X_t = \phi X_{t-1} + \epsilon_t, \quad \text{with } \epsilon_t \sim N(0, \sigma^2).
\]

We learn from Gourieroux [5] that throughout the 1970s the popular time series model was the autoregressive moving average process (ARMA) defined as

\[
X_t = \phi_1 X_{t-1} + \ldots + \phi_k X_{t-k} + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}.
\]

In fact most all linear processes can be represented in the following form of

\[
X_t = \sum_{j=0}^{\infty} c_j \epsilon_{t-j}.
\]

(The following points closely follow Gourieroux [5]) The problem with the above-mentioned models is that they are linear models, which by design restrict the type of dynamics to be approximated. Also the ARMA process is generally applied without imposing a priori constraints on the autoregressive and moving average parameters. These two points, along with the non-linear nature and the conditional variance of financial time series, make the ARMA process a poor fit for, say, foreign currency data.

ARCH and GARCH models, on the other hand, are specific nonlinear time series models, which allow for an exhaustive study of the underlying dynamics. Thus, they provide us with and appropriate framework for studying our foreign currency data.

ARCH and GARCH models, on the other hand, are specific nonlinear time series models, which allow for an exhaustive study of the underlying dynamics. Thus, they provide us with and appropriate framework for studying our foreign currency data.

Gourieroux [5] cautions against using the ARCH and GARCH models as the sole source of information or advice for decision making. He provides the following three limitations to the ARCH models and explains why they should be thought of only as a useful source of supplementary advice for decision making, and should be complemented with more traditional inference methods.

(i) ARCH models are fitted to return series. As we know, the financial decisions depend not only on expected returns and volatilities but also on market shares, on the search for balanced allocations among several categories of assets, and on volumes.
(ii) The ARCH model assumes a rather stable environment and fails to capture irregular phenomena such as crashes, mergers, news effects or threshold effects, opening and closing of the markets, price evolution for an option close to maturity, etc.

(iii) The price evolution is modelled using the common knowledge contained in lagged prices. It does not take into account the possibility of information withheld by individuals or explain how to deal with it.

Since ARCH and GARCH models are employed to forecast future volatility levels, let's now move to a brief discussion on volatility and weighting schemes that leads us directly to our ARCH model.

1.2 Volatility

Hull [7] defines $\sigma_n$ as the volatility of a market variable on day $n$, estimated at the end of day $n-1$. We also note that $\sigma_n^2$ is defined as the variance rate. The common method for modeling this variance rate is to take

$$\sigma_n^2 = \frac{1}{m} \sum_{i=1}^{m} u_{n-i}^2$$

(5)

where $u_i$ is defined to be the logarithm of the relative price $p_i$ from one day to the next,

$$u_i = \log \left( \frac{p_i}{p_{i-1}} \right)$$

(6)

and the mean of the $u_i$'s is assumed to be zero. This assumption is well satisfied by real return series, see e.g. Campbell et al. [2]. We note that by Taylor's formula the $u_i$ in equation (6) are good approximations to the relative difference of the consecutive daily prices $u_i^*$, where

$$u_i^* = \frac{p_i - p_{i-1}}{p_{i-1}}.$$  

(7)

In formulas (6) and (7), $p_i$ is the price of an asset at the end of the $i$th trading day. The $u_i$ are called the log-returns, or simply the returns. We note that the $u_i$ are uncorrelated, i.e., $\text{cov}(u_i, u_{i-1}) = 0$ however, $\text{cov}(u_i^2, u_{i-1}^2) > 0$, see e.g. Campbell et al. [2]. We also notice that in equation (5) an equal weight of $\frac{1}{m}$ is given to all the $u_i$'s. If indeed our goal is to forecast future (next day or more) volatilities, and if we assume that the volatility is subject to change over time, then a weighting scheme that places more importance on recent observations is needed. This is accomplished by simply replacing the $\frac{1}{m}$ weights with positive $\alpha_i$'s, where $0 < \alpha_i < 1$, that give less weight to older observations. It might also be reasonable to assume that there is some long-run average volatility $V$. We add this into our model and apply some weight $\gamma$ to it as well. This yields the model

$$\sigma_n^2 = \gamma V + \sum_{i=1}^{m} \alpha_i u_{n-i}^2$$

(8)
where $\gamma + \sum_{i=1}^{m} \alpha_i$ must sum to 1. Finally if we take $\omega = \gamma V$, we have formed the ARCH(m) model,
\[
\sigma_n^2 = \omega + \sum_{i=1}^{m} \alpha_i u_{n-i}^2,
\] (9)
which considers the previous $m$ observations of $u^2$ in calculating the variance estimate. For our purposes we will focus mostly on ARCH(1) models
\[
\sigma_t^2 = \omega + \alpha u_{t-1}^2,
\] (10)
that consider the single most recent observation of $u^2$. This is reasonable and is often used in practice.

1.3 The EWMA Model

The exponentially weighted moving average (EWMA) model is a natural introductory step to the GARCH(1,1) model. In equation (9) we assigned the $\alpha_i$ weights so that more importance was placed on the more recent observations. In the EWMA model, as we move to an older observation we decrease the weight exponentially. If we assume $i = 1$ to be our most recent observation, by taking $\alpha_{i+1} = \lambda\alpha_i$, where $0 < \lambda < 1$, we can expand equation (9) (ignoring the weighted average volatility $\omega$), substitute, and simplify as shown below to reveal a rather simple model for updating the estimate of the volatility.

\[
\sigma_n^2 = \sum_{i=1}^{m} \alpha_i u_{n-i}^2
\] (11)
\[
= \alpha_1 u_{n-1}^2 + \alpha_2 u_{n-2}^2 + \ldots + \alpha_m u_{n-m}^2
\] (12)
\[
= \alpha_1 u_{n-1}^2 + \lambda\alpha_1 u_{n-2}^2 + \lambda\alpha_2 u_{n-3}^2 + \ldots + \lambda\alpha_{m-1} u_{n-m}^2
\] (13)
\[
= \alpha_1 u_{n-1}^2 + \lambda \sum_{i=1}^{m-1} \alpha_i u_{n-(i+1)}^2
\] (14)
\[
= \alpha_1 u_{n-1}^2 + \lambda \sigma_{n-1}^2
\] (15)
\[
= (1 - \lambda) u_{n-1}^2 + \lambda \sigma_{n-1}^2 \quad \text{(since the weights must sum to 1)}
\] (16)

Similarly we can substitute for $\sigma_{n-1}^2$ and yield
\[
\sigma_n^2 = (1 - \lambda) \sum_{i=1}^{m} \lambda^{i-1} u_{n-i}^2 + \lambda^m \sigma_0^2
\] (17)

which allows us to more clearly see that we are indeed exponentially decreasing the weight at rate $\lambda$ as we move further and further away from recent observations. The advantage of the EWMA model is that large amounts of data and likewise large computations, are not necessary for determining the variance rate $\sigma_n^2$. All we need is the previous estimate and then we use the simple equation (16) to calculate the current volatility estimate.
1.4 The GARCH(1,1) Model

By further assuming that our EWMA model has some long-run average volatility $V$, we weight this $V$ again by $\gamma$ and add it to our model. The result

$$\sigma_n^2 = \gamma V + (1 - \lambda)u_{n-1}^2 + \lambda \sigma_{n-1}^2$$

(18)

must be modified since now our weights must all sum to 1, and therefore the weight for $u_{n-1}^2$ can not be $(1 - \lambda)$. We define $\omega = \gamma V$, $\alpha =$ weight applied to $u_{n-1}^2$, and $\beta =$ weight applied to $\sigma_{n-1}^2$. The subsequent model is our GARCH(1,1)

$$\sigma_n^2 = \omega + \alpha_1 u_{n-1}^2 + \beta \sigma_{n-1}^2,$$

(19)

which is based on only one previous observation of $u^2$ and only one previous value of $\sigma^2$. In a more general GARCH($m,n$) model, we consider the previous $m$ observations of $u^2$ and the previous $n$ values of $\sigma^2$ and define the model by

$$\sigma_t^2 = \omega + \sum_{i=1}^{m} \alpha_i u_{t-i}^2 + \sum_{i=1}^{n} \beta_i \sigma_{t-i}^2.$$ 

(20)

However, for our purposes we are content to utilize solely the GARCH(1,1) model. This model is most commonly used in financial practice.

1.4.1 Mean Reversion

An important advantage of the GARCH(1,1) model over the EWMA model is that the GARCH(1,1) model combines the appealing exponentially declining weights with a property known as mean reversion which is explained below. This is accomplished since the GARCH(1,1) includes the $\omega$ variable that is a weighted measure of the long-run average volatility. The EWMA model does not incorporate this feature and therefore is less appealing for our purposes than the GARCH(1,1).

Hull [7] calculates the expected value of future volatilities to be

$$E[\sigma_{n+k}^2] = V + (\alpha + \beta)^k(\sigma_n^2 - V)$$

(21)

by substituting $\gamma = 1 - \alpha - \beta$ in equation (19), where $\omega = \gamma V$, and taking the expected value of future day $n + k$. From this we can clearly see that if $\alpha + \beta < 1$, our estimate of $\sigma^2$ will get steadily closer and closer to $V$, our long-run average volatility, as we increase $k$, our distance or time of looking into the future. In this context $k$ is called the lead time. This is mean reversion, and we note that for the GARCH(1,1) model to incorporate mean reversion, the condition the $\alpha + \beta < 1$ must be satisfied. If in fact this condition is not met and $\alpha + \beta > 1$, we note that our forecast would distance itself further and further from $V$ as we looked further into the future. This is called mean fleeting by Hull [7], and it is suggested that the EWMA model be used when $\alpha + \beta > 1$. 

10
2 Change Point Problem

Change point theory is a subject that is easily understood and blatantly important in all model fitting, and therefore also has significance in our ARCH and GARCH modeling efforts. In short, we are trying to determine if a change in models, or rather a change in parameters has occurred at some point in our time series. This is of utmost significance since, for example, if there is in fact a change at some point \( t_0 \) and we fail to recognize this, our parameter estimates will be greatly wrong. In other words, we would be fitting one general model and its corresponding parameters to the observed series when in fact two models or two sets of parameters would be appropriate.

In practice, when determining the change point \( t_0 \), one could reasonably begin with points in the data set where a change has most likely occurred with some respect to the source of the data. For example, if a company undertook a change in management or implemented a new marketing technique at time \( t_* \), this would be a plausible starting point to consider when testing for a change in models. For our purposes, we will assume that these facts are not generally available and we will base our techniques and test statistics solely on mathematical and statistical properties and not on a priori information. However, once we have determined that a change has occurred, it is only natural and logical to then search for evidence that supports this claim.

2.1 Generalized Likelihood Ratio

Our first attempt at determining if a change in parameters has occurred involves the familiar generalized likelihood ratio test. As before, we begin with the simple ARCH(1) model and will then generalize for the GARCH(1,1) case.

2.1.1 GLR Statistic for ARCH(1) Model

If we assume that the observations follow an ARCH(1) model as previously defined,

\[
y_t = \sigma_t \epsilon_t, \quad \sigma_t^2 = \omega + \alpha \sigma_{t-1}^2, \quad \epsilon_t \sim \text{iid } N(0, 1), \quad t = 1, \ldots, n,
\]

then this is our null hypothesis. In other words, our null is that there has been no change in parameters, or that only one model is sufficient for our data. Naturally, the alternative hypothesis is that the parameter vector \((\omega, \alpha)\) changes at some unknown time \( t_0 \). Thus, under the alternative, model (22) holds only for \( t \leq t_0 \) and for \( t > t_0 \). The data follow another ARCH(1) model with parameters \( \omega^* \) and \( \alpha^* \) i.e.

\[
y_t = \sigma_t \epsilon_t, \quad \sigma_t^2 = \omega^* + \alpha^* \sigma_{t-1}^2, \quad \epsilon_t \sim \text{iid } N(0, 1),
\]
where $w^* \neq w$ or $\alpha^* \neq \alpha$. Suppose for a moment we know the time $t_0$ when a change in parameter(s) has occurred, then the Generalized Likelihood Ratio (GLR) statistic is defined as

$$
\Lambda_{t_0} = \frac{\text{maximum likelihood under null}}{\text{maximum likelihood if change at } t_0}.
$$

(24)

In order to derive this statistic we consider the conditional density of $y_t$ given $y_1, \ldots, y_{t-1}$, which due to the $N(0, 1)$ distribution of the $e_i$ is simply the density of a $N(0, \sigma^2_t)$ random variable. Thus, the conditional density of $y_t$ given $y_1, \ldots, y_{t-1}$ is

$$
\left[ \frac{2\pi(\omega + \alpha y_{t-1}^2)^{-\frac{1}{2}}}{\omega + \alpha y_{t-1}^2} \right] \exp \left\{ -\frac{1}{2} \cdot \frac{y_t^2}{\omega + \alpha y_{t-1}^2} \right\}
$$

for $t \leq n$ (under the null hypothesis). Now the numerator in our GLR statistic, or the maximum likelihood under the null hypothesis is found to be

$$
\left( \prod_{t=1}^{n} \left[ \frac{2\pi(\omega + \alpha y_{t-1}^2)^{-\frac{1}{2}}}{\omega + \alpha y_{t-1}^2} \right] \frac{1}{2} \sum_{t=1}^{n} \frac{y_t^2}{\omega + \alpha y_{t-1}^2} \right),
$$

(26)

where $\hat{\omega}$ and $\hat{\alpha}$ are the MLE’s. If we define

$$
\hat{\omega} + \hat{\alpha} y_{t-1}^2 = \hat{\sigma}^2_t,
$$

(27)

the above can be simplified and written as

$$
(2\pi)^{-\frac{n}{2}} \left( \prod_{t=1}^{n} \hat{\sigma}^2_t \right)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \sum_{t=1}^{n} \frac{y_t^2}{\hat{\alpha}^2_t} \right\}.
$$

(28)

Let $\tilde{\omega}, \tilde{\alpha}, \tilde{\sigma}$, be the estimates based on $y_1, \ldots, y_{t_0}$ and $\hat{\omega}, \hat{\alpha}, \hat{\sigma}$ the estimates based on $y_{t_0+1}, \ldots, y_n$. If there is a change in parameters at $t_0$, then the maximum likelihood, i.e. the denominator in $\Lambda_{t_0}$ is found by the same method to be

$$
(2\pi)^{-\frac{n}{2}} \left( \prod_{t=1}^{t_0} \tilde{\sigma}^2_t \right)^{-\frac{1}{2}} \left( \prod_{t=t_0+1}^{n} \hat{\sigma}^2_t \right)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \left[ \sum_{t=1}^{t_0} \frac{y_t^2}{\tilde{\alpha}^2_t} + \sum_{t=t_0+1}^{n} \frac{y_t^2}{\hat{\alpha}^2_t} \right] \right\}.
$$

(29)

Thus, by taking equation(26) and equation(29) we can now calculate our GLR statistic $\Lambda_{t_0}$ to be

$$
\Lambda_{t_0} = \left[ \left( \prod_{t=1}^{t_0} \tilde{\sigma}^2_t \right) \left( \prod_{t=t_0+1}^{n} \hat{\sigma}^2_t \right) \right]^\frac{1}{2} \exp \left\{ -\frac{1}{2} \left[ \sum_{t=1}^{n} \frac{y_t^2}{\tilde{\alpha}^2_t} - \sum_{t=1}^{t_0} \frac{y_t^2}{\tilde{\alpha}^2_t} - \sum_{t=t_0+1}^{n} \frac{y_t^2}{\hat{\alpha}^2_t} \right] \right\},
$$

(30)

and so

$$
-2 \log \Lambda_{t_0} = \left[ \sum_{t=1}^{t_0} \left( \log \tilde{\alpha}^2_t - \log \hat{\alpha}^2_t \right) + \sum_{t=t_0+1}^{n} \left( \log \tilde{\alpha}^2_t - \log \hat{\alpha}^2_t \right) \right] + \sum_{t=1}^{n} \frac{y_t^2}{\tilde{\alpha}^2_t} - \sum_{t=1}^{t_0} \frac{y_t^2}{\tilde{\alpha}^2_t} - \sum_{t=t_0+1}^{n} \frac{y_t^2}{\hat{\alpha}^2_t}.
$$

(31)
Since, in fact, we do not know that a change took place at \( t_0 \), we consider the statistic

\[
\Lambda_n^* = \max_{1 \leq k < n} (-2 \log \Lambda_k)
\]

\[
= \max_{1 \leq k < n} \left[ \sum_{t=1}^{n} \left( \frac{y_t^2}{\sigma_t^2} + \log \sigma_t^2 \right) - \sum_{t=1}^{k} \left( \frac{y_t^2}{\sigma_t^2} + \log \sigma_t^2 \right) - \sum_{t=k+1}^{n} \left( \frac{y_t^2}{\sigma_t^2} + \log \sigma_t^2 \right) \right]
\]  

(32)

In practice, to allow for some minimal amount of data in the tails of this statistic, we consider only the main bulk or center, namely \( \max_{T(n) < k < n - T(n)} (-2 \log \Lambda_k) \).

Our goal is to find the empirical distribution function of \( \Lambda_n^* \), particularly the 95th and 99th empirical quantiles. Ideally we would like to consider several ARCH(1) models and several GARCH(1,1) models. This would enable us to determine if the critical values of this test statistic change with respect to the parameters of the model and/or the length of the series. Unfortunately in Splus, the time and/or computer memory necessary to accomplish this appears to be enormous and therefore unfeasible considering the time constraints for this project and our current computer capabilities.

In studying an ARCH(1) model of size 100, we realized that even this simplest and smallest of models still takes several days to generate 1000 \( \Lambda_n^* \) statistics. It is noteworthy to mention that the resulting 95th and 99th empirical quantiles from this model were 13.03476 and 16.568004 respectively. Other empirical quantiles are shown in Table 2 and the code used is supplied in Appendix B.

<table>
<thead>
<tr>
<th>Model ARCH(1)</th>
<th>( \omega=0.1 )</th>
<th>( \alpha=0.4 )</th>
<th>( \beta=0.0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>11.285</td>
<td>11.576</td>
<td>11.910</td>
</tr>
<tr>
<td>0.01</td>
<td>11.576</td>
<td>12.158</td>
<td>12.651</td>
</tr>
<tr>
<td>0.02</td>
<td>12.158</td>
<td>13.035</td>
<td>13.476</td>
</tr>
<tr>
<td>0.03</td>
<td>12.651</td>
<td>13.476</td>
<td>14.116</td>
</tr>
<tr>
<td>0.04</td>
<td>13.035</td>
<td>14.116</td>
<td>15.182</td>
</tr>
<tr>
<td>0.05</td>
<td>13.476</td>
<td>15.182</td>
<td>16.568</td>
</tr>
<tr>
<td>0.06</td>
<td>14.116</td>
<td>15.182</td>
<td>16.568</td>
</tr>
<tr>
<td>0.07</td>
<td>15.182</td>
<td>16.568</td>
<td>16.568</td>
</tr>
<tr>
<td>0.08</td>
<td>16.568</td>
<td>16.568</td>
<td>16.568</td>
</tr>
<tr>
<td>0.09</td>
<td>16.568</td>
<td>16.568</td>
<td>16.568</td>
</tr>
</tbody>
</table>

Table 2: \( \Lambda_n^* \) Empirical quantiles for ARCH(1) model of size 100, based on 1000 iterations using the Splus code given in Appendix B.

Due to the vast amount of time necessary to compute these empirical quantiles when utilizing Splus, another language was considered. Thanks to Gudrun Kokoszka and Gilles Teyssière, C++ code was formulated and supplied resulting in a much faster program for generating ARCH(1) \( \Lambda_n^* \) values. Using this C++ program the same techniques were followed to generate \( \Lambda_n^* \) values. Table 3 shows the empirical quantiles which were taken from 1000 generated \( \Lambda_n^* \) values for each of the differing models.
Immediately we are able to identify that the critical values do change depending on the size of the data set. This dependence on $n$ was expected since the statistic increases roughly at the rate of $\ln(\ln(n))$ even if the observations are iid, see Csörgő and Horváth [4]. We also notice that the empirical quantiles found when using the C++ code are different from those found when applying the Splus code. There are some reasonable explanations for this occurrence. One explanation is that within the Splus code we are simply taking the squared residuals as they are estimated and produced by the garch module. This may be a different method of estimation, and therefore may return different values, than that of the C++ code.

Since there has been no theoretical work previously performed on this particular statistic, we were unaware as to if the empirical quantiles would be dependent upon the model. The test statistic $\Lambda^*_n$ depends on the estimated squared residuals, $\epsilon_t^2$. We were hoping that these squared residuals for ARCH(1) (which is a non-linear model), would behave like the iid innovations $\sigma_t^2$ no matter what the model. However we see from our results that this is not the case. The upper

<table>
<thead>
<tr>
<th>ARCH(1) Models</th>
<th>Model 1 $\omega = 0.1, \alpha = 0.4$</th>
<th>Model 2 $\omega = 0.35, \alpha = 0.1$</th>
<th>Model 3 $\omega = 0.2, \alpha = 0.5$</th>
<th>Model 4 $\omega = 0.6, \alpha = 0.3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>90th %tile</td>
<td>$n=100$ 9.12488</td>
<td>$n=100$ 11.1230</td>
<td>$n=100$ 10.4483</td>
<td>$n=100$ 11.5770</td>
</tr>
<tr>
<td></td>
<td>$n=250$ 12.2281</td>
<td>$n=250$ 13.2858</td>
<td>$n=250$ 13.0612</td>
<td>$n=250$ 13.9951</td>
</tr>
<tr>
<td></td>
<td>$n=500$ 13.4633</td>
<td>$n=500$ 15.1689</td>
<td>$n=500$ 14.3984</td>
<td>$n=500$ 15.6463</td>
</tr>
<tr>
<td>95th %tile</td>
<td>$n=100$ 10.3663</td>
<td>$n=100$ 12.2807</td>
<td>$n=100$ 11.8901</td>
<td>$n=100$ 12.7137</td>
</tr>
<tr>
<td></td>
<td>$n=250$ 14.1810</td>
<td>$n=250$ 14.4058</td>
<td>$n=250$ 14.7022</td>
<td>$n=250$ 15.3497</td>
</tr>
<tr>
<td></td>
<td>$n=500$ 15.1274</td>
<td>$n=500$ 16.7274</td>
<td>$n=500$ 15.7348</td>
<td>$n=500$ 16.9788</td>
</tr>
<tr>
<td>96th %tile</td>
<td>$n=100$ 10.7478</td>
<td>$n=100$ 12.7039</td>
<td>$n=100$ 12.3217</td>
<td>$n=100$ 13.3241</td>
</tr>
<tr>
<td></td>
<td>$n=250$ 14.6324</td>
<td>$n=250$ 15.0453</td>
<td>$n=250$ 15.0085</td>
<td>$n=250$ 15.7193</td>
</tr>
<tr>
<td></td>
<td>$n=500$ 15.6278</td>
<td>$n=500$ 17.2354</td>
<td>$n=500$ 16.1726</td>
<td>$n=500$ 17.5064</td>
</tr>
<tr>
<td>97th %tile</td>
<td>$n=100$ 11.6853</td>
<td>$n=100$ 13.3043</td>
<td>$n=100$ 12.8607</td>
<td>$n=100$ 13.8454</td>
</tr>
<tr>
<td></td>
<td>$n=250$ 14.8787</td>
<td>$n=250$ 15.4774</td>
<td>$n=250$ 15.5600</td>
<td>$n=250$ 16.3205</td>
</tr>
<tr>
<td></td>
<td>$n=500$ 16.1695</td>
<td>$n=500$ 17.8432</td>
<td>$n=500$ 16.5481</td>
<td>$n=500$ 17.9789</td>
</tr>
<tr>
<td>98th %tile</td>
<td>$n=100$ 12.1168</td>
<td>$n=100$ 14.0085</td>
<td>$n=100$ 13.8339</td>
<td>$n=100$ 14.4260</td>
</tr>
<tr>
<td></td>
<td>$n=250$ 15.9811</td>
<td>$n=250$ 16.2162</td>
<td>$n=250$ 16.4239</td>
<td>$n=250$ 17.2585</td>
</tr>
<tr>
<td></td>
<td>$n=500$ 16.8367</td>
<td>$n=500$ 18.2985</td>
<td>$n=500$ 17.8619</td>
<td>$n=500$ 18.8854</td>
</tr>
<tr>
<td>99th %tile</td>
<td>$n=100$ 13.0512</td>
<td>$n=100$ 15.4125</td>
<td>$n=100$ 15.1239</td>
<td>$n=100$ 16.0472</td>
</tr>
<tr>
<td></td>
<td>$n=250$ 16.9449</td>
<td>$n=250$ 17.6981</td>
<td>$n=250$ 18.3362</td>
<td>$n=250$ 18.6434</td>
</tr>
<tr>
<td></td>
<td>$n=500$ 17.9893</td>
<td>$n=500$ 19.3676</td>
<td>$n=500$ 19.4667</td>
<td>$n=500$ 20.0680</td>
</tr>
</tbody>
</table>

Table 3: $\Lambda^*_n$ Empirical quantiles for ARCH(1) model, based on 1000 iterations using the C++ program given in Appendix C
quantiles of the distribution of $\Lambda_n^*$ do depend on the underlying model, making it impossible to find universal critical values for a given sample size. This situation is similar to the one stated in Horváth et al. [6] for the empirical process of the squared residuals, where the limiting distribution explicitly depends on the model parameters.

Due to the dependence on the model, one would have to rely upon bootstrap or Monte Carlo methods for this GLR test. In order to accomplish this, one would perform the following steps.

(i) Calculate from the data the observed value of the statistic $\lambda_n^*$.

(ii) Generate $N$ series of length $n$, using the estimated values from the original data as the parameters of these generated series.

(iii) Calculate the corresponding $N$ bootstrap values of $\Lambda_n^*$.

(iv) Determine the bootstrap $p$-value defined as the proportion of the $N$ values of $\Lambda_n^*$ calculated in the previous step that are greater than the $\lambda_n^*$ calculated in step (i). Similarly one could consider the empirical distribution of the $N$ values of $\Lambda_n^*$ in order to determine critical values.

One would need to determine the power of this procedure by repeating the above steps and considering the percentage of times the $\lambda_n^*$ observed value either exceeds or fails to exceed the calculated critical values.

Given the available resources and our limited time frame, an endeavor such as this would greatly exceed the scope of this project. Nevertheless, before we examine other change-point test methods, we demonstrate in the next section how to derive a similar GLR test for a GARCH(1,1) model.

2.1.2 GLR Statistic for GARCH(1,1) Model

Similar steps can be taken to compute the statistic $\Lambda_n^*$ for the GARCH(1,1) model. Naturally we begin with the previously defined GARCH(1,1) model

\[ \sigma_t^2 = \omega + \alpha_1 r_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \]  

and this is our null hypothesis. The alternative hypothesis is that the parameter vector $(\omega, \alpha, \beta)$ changes at some unknown time $t_0$. Again, under the alternative hypothesis, equation(34) holds only for $t \leq t_0$ and for $t > t_0$.

We now define another GARCH(1,1) model with parameters $\omega^*$, $\alpha^*$, and $\beta^*$ to be

\[ y_t = \sigma_t \epsilon_t, \quad \sigma_t^2 = \omega^* + \alpha^* y_{t-1}^2 + \beta^* \sigma_{t-1}^2, \quad \epsilon_t \sim \text{iid } N(0,1), \quad t = 1, \ldots, n., \]  

15
where $\omega^* \neq \omega$, $\alpha^* \neq \alpha$, or $\beta^* \neq \beta$. Again, suppose for a moment we know the time $t_0$ when a change in parameter(s) has occurred, then the Generalized Likelihood Ratio (GLR) statistic is still defined as

$$\Lambda_{t_0} = \frac{\text{maximum likelihood under null}}{\text{maximum likelihood if change at } t_0}. \quad (36)$$

In order to derive this statistic we consider the conditional density of $y_t$ given $y_1, \ldots, y_{t-1}$, which due to the $N(0,1)$ distribution of the $\epsilon_i$ is still the density of a $N(0, \sigma_i^2)$ random variable. Thus, the conditional density of $y_t$ given $y_1, \ldots, y_{t-1}$ is now

$$\left[2\pi(\omega + \alpha y_{t-1}^2 + \beta \sigma_i^2)\right]^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \cdot \frac{y_t^2}{\omega + \alpha y_{t-1}^2 + \beta \sigma_i^2} \right\}, \quad (37)$$

for $t \leq n$ (under the null hypothesis).

Now the numerator in our GLR statistic, or the maximum likelihood under the null hypothesis is found to be

$$\left(\prod_{t=1}^{n} \left[2\pi(\omega + \alpha y_{t-1}^2 + \beta \sigma_i^2)\right]^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \cdot \sum_{t=1}^{n} \frac{y_t^2}{\omega + \alpha y_{t-1}^2 + \beta \sigma_i^2} \right\}, \quad (38)$$

where $\hat{\omega}$, $\hat{\alpha}$, and $\hat{\beta}$ are the MLE’s. We now observe that $\hat{\omega} + \hat{\alpha} y_{t-1}^2 + \hat{\beta} \sigma_i^2 = \hat{\sigma}_i^2$, so the above can be simplified and written as

$$(2\pi)^{-\frac{1}{2}} \left(\prod_{t=1}^{n} \hat{\sigma}_i^2\right)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \cdot \sum_{t=1}^{n} \frac{y_t^2}{\hat{\sigma}_i^2} \right\}, \quad (39)$$

From here we can easily see that this is exactly the same as equation (28), and therefore the GLR statistic $\Lambda_n^*$ for GARCH(1,1) models will be the same as for ARCH(1) models, namely

$$\Lambda_n^* = \max_{1 \leq k < n} \left( -2 \log \Lambda_k \right) \quad (40)$$

$$= \max_{1 \leq k < n} \left[ \sum_{t=1}^{n} \left( \frac{y_t^2}{\hat{\sigma}_t^2} + \log \hat{\sigma}_t^2 \right) - \sum_{t=1}^{k} \left( \frac{y_t^2}{\hat{\sigma}_t^2} + \log \hat{\sigma}_t^2 \right) - \sum_{t=k+1}^{n} \left( \frac{y_t^2}{\hat{\sigma}_t^2} + \log \hat{\sigma}_t^2 \right) \right]. \quad (41)$$

Again we note that the time necessary for deriving the empirical quantiles of even the smallest GARCH(1,1) model of size 100 would take several days and therefore would extend beyond the constraints of this project. With such an impediment to this first approach, we are forced to consider another method in which we might obtain similar information about change-points, but most likely with less power.
3 Change-Point Tests Based on the Sequential Empirical Process

3.1 Kolmogorov-Smirnov Type Statistic

In time-series modeling we are often constrained due to the dependent and conditional nature of the \( y_i \) observations. However, within the ARCH and GARCH modeling techniques we assume that the \( \epsilon_i \) are independent and identically distributed. With this in mind, we can utilize tests based upon the empirical distribution function of the \( \epsilon_i^2 \). Since we do not know the actual squared errors \( \epsilon_i^2 \), we are forced to estimate them by the squared residuals \( \hat{\epsilon}_i^2 \) defined by

\[
\hat{\epsilon}_i^2 = \frac{y_i^2}{\hat{\sigma}^2_i}.
\]  

(42)

If, as previously discussed, our parameter estimates for \( \omega, \alpha \) and \( \beta \) are fairly accurate, then our estimated \( \hat{\sigma}^2_i \) will also be accurate and it follows that \( \hat{\epsilon}_i^2 \) will likewise be a good estimate. We now construct several test statistics based on the \( \hat{\epsilon}_i^2 \). For \( t \in (0, \infty) \) we define the empirical cumulative distribution function (ecdf) of the \( \hat{\epsilon}_i^2 \) for the first \( k \) observations \( (\hat{F}_k(t)) \), and the ecdf for the last \( n-k \) observations \( (\hat{F}_n^*(t)) \) to be

\[
\hat{F}_k(t) = \frac{1}{k} \# \{ i \leq k : \hat{\epsilon}_i^2 \leq t \} 
\]  

(43)

\[
\hat{F}_n^*(t) = \frac{1}{n-k} \# \{ i > k : \hat{\epsilon}_i^2 \leq t \}.
\]  

(44)

With \( \hat{F}_k(t) \) and \( \hat{F}_n^*(t) \) defined we can now create a test statistic by taking the difference of the two ecdfs and adjusting for the position of \( k \):

\[
\hat{T}_n(k,t) = \sqrt{n} \cdot \frac{k}{n} \left( 1 - \frac{k}{n} \right) | \hat{F}_k(t) - \hat{F}_n^*(t) |
\]  

(45)

\[
\hat{M}_n(t) = \max_{1 \leq k \leq n} | \hat{T}_n(k,t) |
\]  

(46)

\[
\hat{M}_n = \sup_{0 \leq t \leq \infty} | \hat{M}_n(t) |
\]  

(47)

If the two pieces of our data, namely the first \( k \) observations compared to the last \( n-k \) observations, are similar with respect to their parameters, we would expect the difference in the ecdfs of the squared residuals \( \hat{\epsilon}_i^2 \), as well as the statistic \( \hat{T}_n(k,t) \) to be small. However, when there is a change or difference in the two parts of our data, there would most likely be a difference in the ecdfs of the squared residuals, and therefore we would expect our statistic \( \hat{T}_n(k,t) \) to be large. By taking the maximum of this statistic with respect to \( k \) and the supremum over all \( t \), we can determine if a change in our model has occurred.

The statistic \( \hat{M}_n \) is a generalization of the Kolmogorov-Smirnov statistic and is discussed by Picard [9]. The asymptotic distribution of \( \hat{M}_n \) under the null hypothesis is known, and we are presumably given critical values for this test statistic by Picard [9] on page 845. Since this table is
not clearly referenced in his paper, we needed a rough verification by running simulations of this test statistic using $N(0, 1)$ random variables $y_i$. This was performed and the resulting figures are indeed similar to those shown in Picard’s [9] table. Table 4 shows the evaluation of the threshold $C$ for different values of the level $\alpha$, and the Splus code that was used is supplied in Appendix C. We note that $\alpha = P(\hat{M}_n > C)$. A quick comparison of this table to Picard’s verifies that the values given in his paper are indeed the critical values for $\hat{M}_n$. The threshold values for our generated table are only based upon 100 replications of the test statistic. Due to this fact it is of no surprise that we encounter slightly different values than Picard due to chance error.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>0.00</th>
<th>0.01</th>
<th>0.02</th>
<th>0.03</th>
<th>0.04</th>
<th>0.05</th>
<th>0.06</th>
<th>0.07</th>
<th>0.08</th>
<th>0.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.856</td>
<td>0.820</td>
<td>0.815</td>
<td>0.799</td>
<td>0.768</td>
<td>0.758</td>
<td>0.752</td>
<td>0.751</td>
<td>0.740</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.740</td>
<td>0.736</td>
<td>0.735</td>
<td>0.731</td>
<td>0.720</td>
<td>0.708</td>
<td>0.702</td>
<td>0.692</td>
<td>0.690</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.678</td>
<td>0.668</td>
<td>0.657</td>
<td>0.650</td>
<td>0.640</td>
<td>0.627</td>
<td>0.612</td>
<td>0.610</td>
<td>0.604</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.600</td>
<td>0.600</td>
<td>0.596</td>
<td>0.596</td>
<td>0.595</td>
<td>0.582</td>
<td>0.580</td>
<td>0.576</td>
<td>0.571</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.566</td>
<td>0.556</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
<td>0.540</td>
<td>0.539</td>
<td>0.525</td>
<td>0.524</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.520</td>
<td>0.520</td>
<td>0.518</td>
<td>0.511</td>
<td>0.509</td>
<td>0.508</td>
<td>0.502</td>
<td>0.496</td>
<td>0.492</td>
<td>0.484</td>
</tr>
<tr>
<td>0.6</td>
<td>0.480</td>
<td>0.480</td>
<td>0.477</td>
<td>0.477</td>
<td>0.476</td>
<td>0.466</td>
<td>0.464</td>
<td>0.460</td>
<td>0.460</td>
<td>0.454</td>
</tr>
<tr>
<td>0.7</td>
<td>0.452</td>
<td>0.450</td>
<td>0.450</td>
<td>0.450</td>
<td>0.449</td>
<td>0.448</td>
<td>0.447</td>
<td>0.444</td>
<td>0.444</td>
<td>0.441</td>
</tr>
<tr>
<td>0.8</td>
<td>0.436</td>
<td>0.435</td>
<td>0.432</td>
<td>0.430</td>
<td>0.428</td>
<td>0.411</td>
<td>0.408</td>
<td>0.406</td>
<td>0.404</td>
<td>0.404</td>
</tr>
<tr>
<td>0.9</td>
<td>0.400</td>
<td>0.400</td>
<td>0.393</td>
<td>0.390</td>
<td>0.386</td>
<td>0.379</td>
<td>0.376</td>
<td>0.370</td>
<td>0.368</td>
<td>0.360</td>
</tr>
</tbody>
</table>

Table 4: Generated threshold $C$ values for different values of the level $\alpha$, based on 100 replications

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>0.00</th>
<th>0.01</th>
<th>0.02</th>
<th>0.03</th>
<th>0.04</th>
<th>0.05</th>
<th>0.06</th>
<th>0.07</th>
<th>0.08</th>
<th>0.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.874</td>
<td>0.823</td>
<td>0.806</td>
<td>0.789</td>
<td>0.772</td>
<td>0.756</td>
<td>0.743</td>
<td>0.733</td>
<td>0.724</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.715</td>
<td>0.706</td>
<td>0.698</td>
<td>0.690</td>
<td>0.682</td>
<td>0.675</td>
<td>0.668</td>
<td>0.662</td>
<td>0.656</td>
<td>0.650</td>
</tr>
<tr>
<td>0.2</td>
<td>0.645</td>
<td>0.640</td>
<td>0.636</td>
<td>0.632</td>
<td>0.628</td>
<td>0.623</td>
<td>0.616</td>
<td>0.611</td>
<td>0.607</td>
<td>0.603</td>
</tr>
<tr>
<td>0.3</td>
<td>0.600</td>
<td>0.597</td>
<td>0.594</td>
<td>0.591</td>
<td>0.587</td>
<td>0.584</td>
<td>0.580</td>
<td>0.576</td>
<td>0.573</td>
<td>0.570</td>
</tr>
<tr>
<td>0.4</td>
<td>0.567</td>
<td>0.564</td>
<td>0.561</td>
<td>0.558</td>
<td>0.555</td>
<td>0.552</td>
<td>0.549</td>
<td>0.546</td>
<td>0.543</td>
<td>0.540</td>
</tr>
<tr>
<td>0.5</td>
<td>0.538</td>
<td>0.536</td>
<td>0.533</td>
<td>0.531</td>
<td>0.529</td>
<td>0.526</td>
<td>0.523</td>
<td>0.521</td>
<td>0.518</td>
<td>0.515</td>
</tr>
<tr>
<td>0.6</td>
<td>0.512</td>
<td>0.509</td>
<td>0.506</td>
<td>0.503</td>
<td>0.499</td>
<td>0.496</td>
<td>0.493</td>
<td>0.490</td>
<td>0.487</td>
<td>0.484</td>
</tr>
<tr>
<td>0.7</td>
<td>0.481</td>
<td>0.478</td>
<td>0.475</td>
<td>0.472</td>
<td>0.469</td>
<td>0.466</td>
<td>0.463</td>
<td>0.461</td>
<td>0.458</td>
<td>0.456</td>
</tr>
<tr>
<td>0.8</td>
<td>0.453</td>
<td>0.450</td>
<td>0.447</td>
<td>0.444</td>
<td>0.441</td>
<td>0.438</td>
<td>0.435</td>
<td>0.431</td>
<td>0.427</td>
<td>0.424</td>
</tr>
<tr>
<td>0.9</td>
<td>0.420</td>
<td>0.416</td>
<td>0.412</td>
<td>0.408</td>
<td>0.404</td>
<td>0.400</td>
<td>0.395</td>
<td>0.387</td>
<td>0.373</td>
<td>0.356</td>
</tr>
</tbody>
</table>

Table 5: Picard’s threshold $C$ values for different values of the level $\alpha$
Remark: An exact and often faster way to compute $\hat{M}_n$ is as follows. Observe that $\hat{F}_k(t)$ is for every fixed $k$ a jump function with jumps at $\varepsilon_i^2$. Thus it may be useless to compare or derive $\hat{T}_n(k,t)$ for all possible $t$ when in fact nothing is occurring except at the points where $t = \varepsilon_i^2$. Therefore, instead of using all $t \in (0, \infty)$, we can simply substitute $t$ for $\varepsilon_i^2$ as follows:

$$\sup_{0 \leq t \leq \infty} \hat{M}_n(t) = \max_{1 \leq j \leq n} \hat{M}_n(\varepsilon_i^2).$$

Therefore instead of equation (47), we have

$$\hat{M}_n = \max_{1 \leq k \leq n} \max_{1 \leq j \leq n} |\hat{T}_n(k,\varepsilon_j^2)|.$$

### 3.2 Cramer-von Mises Type Statistic

Similar to the previous $\hat{M}_n$, the Cramer-von Mises Type Statistic utilizes the empirical distribution function of the squared residuals, but instead of considering the maximum of the absolute value of $\hat{T}_n(k,\varepsilon_j^2)$, we consider the sum of the squared $\hat{T}_n(k,\varepsilon_j^2)$. Blum et al. [1] tabulated the distribution function of the statistic

$$B = \int_0^1 \int_0^1 T^2(s,u)du ds,$$  \hspace{1cm} (50)

where $T$ is the “tied-down” Kiefer process, i.e. a Gaussian field on $[0, 1] \times [0, 1]$ with the covariance structure

$$E[T(s,u)] = 0$$

$$E[T(x,y)T(s,u)] = (x \wedge s - xs)(y \wedge u - yu),$$

where $a \wedge b = \min(a, b)$.

We now introduce the random field

$$\tilde{W}_n(s,t) = \hat{T}_n([us],t),$$  \hspace{1cm} (51)

where $[a]$ denotes the integer part of $[a]$. Horváth et al. [6] showed that $\tilde{W}_n(s,t)$ converges in an appropriate Skorokhod space to $T(s,F(t))$, where $F$ is the cdf of $\varepsilon_i^2$. We denote by $\hat{F}$ the empirical distribution function of $\varepsilon_1^2, \ldots, \varepsilon_n^2$, i.e. $\hat{F}$ places mass $\frac{1}{n}$ at each $\varepsilon_i^2$.

We may expect that the distribution of

$$\hat{B}_n = \int_0^1 \int_0^\infty [\hat{T}_n([us],t)]^2 d\hat{F}(t) ds$$  \hspace{1cm} (52)

will, for large $n$, be well approximated by the distribution of $B$. 

19
To give a rough justification to this claim observe that

\[
\hat{B}_n = \int_0^1 \left[ \int_0^\infty [\hat{W}_n(s, t)] d\hat{F}(t) \right] ds \\
\approx \int_0^1 \left[ \int_0^\infty [\hat{T}(s, F(t))]^2 dF(t) \right] ds \\
= \int_0^1 \left[ \int_0^1 [T(s, v)]^2 dv \right] ds \quad (v = F(t)) \\
= B.
\]

We now explain how to calculate \( \hat{B}_n \). Observe that for any function \( g \)

\[
\int_0^\infty g(t) d\hat{F}(t) = \frac{1}{n} \sum_{j=1}^n g(\hat{\epsilon}_j^2)
\]

and

\[
\int_0^\infty g([us]) ds = \frac{1}{n} \sum_{k=1}^n g(k).
\]

Using the above identities, we obtain

\[
\hat{B}_n = \int_0^1 \left\{ \frac{1}{n} \sum_{j=1}^n [\hat{T}_n([us], \hat{\epsilon}_j^2)]^2 \right\} ds \\
= \frac{1}{n^2} \sum_{k=1}^n \sum_{i=1}^n [\hat{T}_n(k, \hat{\epsilon}_i^2)]^2.
\]

The \( p \)-values and approximate critical values for \( \frac{1}{2} \pi^4 \hat{B}_n \) can be found on p. 497 of Blum et al. [1]. By comparing equation (56) to the previous equation (49), we can see that the two equations are basically estimating the same thing but in slightly different fashions. Again we can construct a rough or approximate check of these table values using \( N(0, 1) \) data and considering the empirical quantiles. Tables 6 and 7 show some of the empirical quantiles, generated from only 100 replications of the statistic, compared to Blum’s actual table values. We expect there to be some amount of chance error in our generated empirical quantiles, and furthermore the set up of Blum’s table also limits the accuracy of our comparisons. Notwithstanding these facts, we conclude that our method for generating this test statistic \( \hat{B}_n \) is correct and that the tables are indeed giving us the critical values of \( \frac{1}{2} \pi^4 \hat{B}_n \).

\[\Box\]

4 Analysis of Foreign Currency Data

4.1 Returns on the DM Exchange Rate

Our first step, and the first step in any time-series analysis, is to plot the data. The top of Figure 1 shows the 6630 logarithmic daily returns on the DM exchange rate from 24 August 1971 to 21 January 1997.
Table 6: Checking Blum's values, $F(y) = \lim_{n \to \infty} P_{H_0}\{\frac{1}{2} \pi^4 \hat{B}_n \leq y\}$, generated values based on 100 replications

<table>
<thead>
<tr>
<th>Generated Values</th>
<th>Blum’s Values</th>
<th>Generated Values</th>
<th>Blum’s Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$F(y)$</td>
<td>$y$</td>
<td>$F(y)$</td>
</tr>
<tr>
<td>0.4507949</td>
<td>0.010</td>
<td>0.40</td>
<td>0.00086</td>
</tr>
<tr>
<td>0.6039466</td>
<td>0.030</td>
<td>0.60</td>
<td>0.04867</td>
</tr>
<tr>
<td>0.6534665</td>
<td>0.080</td>
<td>0.65</td>
<td>0.07899</td>
</tr>
<tr>
<td>0.6928499</td>
<td>0.120</td>
<td>0.70</td>
<td>0.11594</td>
</tr>
<tr>
<td>0.7455490</td>
<td>0.160</td>
<td>0.75</td>
<td>0.15784</td>
</tr>
<tr>
<td>0.7983408</td>
<td>0.200</td>
<td>0.80</td>
<td>0.20293</td>
</tr>
<tr>
<td>0.9470027</td>
<td>0.290</td>
<td>0.95</td>
<td>0.34267</td>
</tr>
<tr>
<td>1.0050624</td>
<td>0.320</td>
<td>1.00</td>
<td>0.38730</td>
</tr>
<tr>
<td>1.0593056</td>
<td>0.380</td>
<td>1.05</td>
<td>0.42994</td>
</tr>
<tr>
<td>1.1069830</td>
<td>0.420</td>
<td>1.10</td>
<td>0.47027</td>
</tr>
</tbody>
</table>

Table 7: Checking critical values of Blum, generated values based on 100 replications

<table>
<thead>
<tr>
<th>Generated Values</th>
<th>Blum’s Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$F^{-1}(p)$</td>
</tr>
<tr>
<td>0.90</td>
<td>2.2273563</td>
</tr>
<tr>
<td>0.95</td>
<td>2.5514129</td>
</tr>
<tr>
<td>0.98</td>
<td>3.3533871</td>
</tr>
<tr>
<td>0.99</td>
<td>3.7377366</td>
</tr>
</tbody>
</table>

We note that this is the same data which was examined by Kokoszka and Leipus [8], however our objective is quite different from that of Kokoszka and Leipus. We are determining whether or not a change has occurred, and if it has we will then estimate when or where this change took place. Kokoszka and Leipus, on the other hand, without formally checking it by means of a statistical test, assumed in advance that a change had occurred, then went about estimating the change-point.

Since we have previously determined and defined several test statistics for determining change-points, it is only natural that we now check to see if these tests indicate a change-point. Again, we discover to be somewhat constrained by the enormity of the data set. Assuming we had enough computer memory, which we don’t, it would take approximately one month each to calculate the $\hat{M}_n$ and $\hat{B}_n$ statistics for a data set of this size using Splus. One way to work around

21
this is to consider the weekly returns in lieu of the daily returns. By doing this we can still use the entire data set, and for the size of the data we would expect the weekly returns to show the same basic trends and patterns as would the daily returns.

The weekly returns are found by simply adding the daily returns from days Monday through Friday for each of the 1,326 weeks in our data set. This reduces our data to a size where we can calculate the change-point test statistics in a more timely manner. A plot of the weekly returns is given in the bottom half of Figure 1, and we can visually see the same basic trends and patterns for this plot as seen in the Daily Returns.
The following Figure 2 shows a time series plot of the statistic $\hat{M}_n$ determined at each value $k$. The supremum of these values is 1.656859 which greatly exceeds even the $\alpha=0.01$ critical value of 0.874 previously shown in Table 5. Therefore the $\hat{M}_n$ statistic is suggesting that a change-point does in fact exist in our model, and we can determine where this change-point occurred by observing from what value $k$ this supremum came.

We see from Figure 2 that the peak is around $k = 450$, which corresponds to week 450 of the DM exchange rate. A closer inspection indicates that the peak is actually at week 443, or in terms of our data, around the year 1979. This is in harmony with the findings presented by Kokoszka and Leipus [8]. It is also interesting to note that in 1979 the European Monetary System (EMS) was introduced. This supports, or rather gives an explanation, as to the reason for why there might have been a change in models in our data at this particular time. We continue our change point tests by segmenting the data set into two pieces, namely weeks 1-443 and weeks 444-1326. By then calculating the $\hat{M}_n$ test statistic on these pieces we check for sub-level change points that may have not been evident when considering the entire data set as a whole. We continue in this manner until we have reached a point where the segments are found to contain no additional change points.

A high-level of significance is considered necessary in order to reject our previously stated
null hypothesis of no change-points. Therefore a strict significance level of \( \alpha = 0.05 \) was not necessarily followed. In fact if a \( p \)-value was found to be around 0.05, this was not considered enough evidence for rejection.

Finally we can estimate the parameters of these segments, assuming that they follow a GARCH(1,1) model. Table 8 summarizes these actions as performed on the German Mark data set.

| \( \hat{M}_n \) Statistic on German Mark Data Beginning with Entire Data Set |
|-----------------|-----------------|-----------------|-----------------|
| Week            | \( \hat{M}_n \) Value | \( p \)-value     | \( \hat{M}_n \) Value | \( p \)-value     | \( \hat{M}_n \) Value | \( p \)-value     |
| 443             | 1.656859         | \( p < 0.01 \)    | 85              | **0.850058**      | \( 0.01 < p < 0.02 \) | Not Sig.         | 0.658419         |
| 0.03 < \( p < 0.04 \) | 0.799861         | **0.586988**      | 247             | **0.499534**      | \( 0.63 < p < 0.64 \) | Not Sig.         | 0.708455         |
| \( p \approx 0.34 \) | 0.708455         | **0.499534**      | 443             | 0.0001203         | \( 0.009423 \)     | **0.499534**      | \( 0.06903 \)    |
| \( 0.000012875 \) | 0.00001585       | **0.499534**      | \( 0.06903 \)   | 0.49830           | \( 0.99710 \)     | **0.499534**      | \( 0.86650 \)    |

Table 8: Segmentation Summary of German Mark Data Using \( \hat{M}_n \) Statistic

Remark: It is noteworthy to mention that for certain segments of the data, Splus reported a warning message when performing the GARCH(1,1) modeling operation. The message stated “The estimated asymptotic variance is not well-defined.” The exact impact of this on our change-point testing procedures is unclear. The statistics which yielded such warnings are denoted with asterisks, e.g. **0.499534**.

Similarly we can calculate the test statistic \( \hat{B}_n \) in order to further validate our change-point findings. Since \( \hat{B}_n \) is essentially determining the same type of change as was \( \hat{M}_n \), we can expect to see similar results. In fact the value of \( \hat{B}_n \) is calculated to be 28.94558. Again, this results in a \( p \)-value of approximately 0.00, and we conclude that a change-point is present. The following Figure
3 shows a plot of the values obtained from the first summation, $\sum_{i=1}^{n}(\hat{T}_n(k, \hat{\varepsilon}_i^2))^2$, of $\hat{B}_n$. These values provide us with an insight as to when the change-point was most likely to have occurred.

![Figure 3: $\hat{B}_n$ Values vs. Time (Weeks) for the German Mark Data](image)

As expected we observe a similar pattern and we notice that the values also peak at week 443. Since a change-point was detected at week 443, we then segment the data down into weeks 1-443 and weeks 444-1326 then calculate the $\hat{B}_n$ test statistic for each segment. Table 9 shows the $\hat{B}_n$ values obtained from these operations and unlike the previous $\hat{M}_n$ tests, both of these are found to be insignificant, or rather, are not extreme enough to conclude that further change-points exist. The last operation is to again estimate the parameters of these two segments.

It appears that in the German Mark data set, the $\hat{M}_n$ statistic is a little more powerful in detecting changes than is the $\hat{B}_n$ statistic. We note that the $\hat{M}_n$ test statistic found three change-points while the $\hat{B}_n$ statistic only found one. Naturally, we can not be certain that either of these results are truly detecting the change-points that may or may not really exist in the series. However, as we look across the final segments of the data and the estimated parameters of those segments, it is evident that the estimated models of those different pieces are quite different. Assuming that our parameter estimation is accurate, this would be evidence that indeed there exist change-points within the series.
\begin{tabular}{|c|c|c|}
\hline
& \textbf{\(\hat{B}_n\) Statistic on German Mark Data Beginning with Entire Data Set} & \\
\hline
Week & 443 & \\
\(\hat{B}_n\) Value & 15.27392 & \\
p-value & \(p < 0.01\) & \\
\hline
Week & Not Sig. & Not Sig. & \\
\(\hat{B}_n\) Value & **2.735305** & 1.984714 & \\
p-value & \(0.0560 < p < 0.0595\) & \(0.1453 < p < 0.1554\) & \\
\hline
Final Segments and Corresponding GARCH(1,1) Parameter Estimates & \\
Weeks & 1-443 & 444-1326 & \\
\(\omega\) & 0.000008308 & 0.00001561 & \\
\(\alpha\) & 0.27310 & 0.06894 & \\
\(\beta\) & 0.72970 & 0.86750 & \\
\hline
\end{tabular}

Table 9: Segmentation Summary of German Mark Data Using \(\hat{B}_n\) Statistic

4.2 Returns on the British Pound Exchange Rate

The British Pound exchange rate data set spans the same time frame as previously explained for the DM exchange rate data set. Again, the size of the resulting data set is too large for the scope of this project, and we are forced to transform the data into weekly returns. The following Figure 4 shows the daily returns and the weekly returns for the British Pound exchange rate data. Similar to the German Mark data, after performing the conversion from daily returns to weekly returns, one can still visually discern the patterns and trends that were evident before making the change.

As previously performed on the German Mark data, we conduct the change-point tests using the \(\hat{M}_n\) statistic and segment the data according to the determined change-points. We continue segmenting and testing until we are left with pieces that are determined to be individually homogeneous in model, then we estimate the parameters of those models. The following table 10 summarizes these operations as performed on the British Pound Data. Again we find several change-points when using the \(\hat{M}_n\) statistic, and we also notice a distinct difference in the estimated model parameters as we move from one segment to the next.

Performing the same operations while using the \(\hat{B}_n\) statistic yields quite different results. Table 11 summarizes these results and we note that only one change point was found. Also, the change-point was found to be at week 403 which does not directly correspond to any of the change-points found using the \(\hat{M}_n\) statistic. This is, however, consistent with the findings on the German Mark data. Again we notice that the \(\hat{M}_n\) statistic seems to be slightly more sensitive to detecting change-points compared to the \(\hat{B}_n\) statistic. We also see a visible difference in the estimated model parameters.
parameters of the final segments. Again, assuming that our estimation is accurate, this is evidence that change-points truly do exist in the series.
Table 10: Segmentation Summary of Pound Data Using $\tilde{M}_n$ Statistic

### 5 Checking the Power of the Change-Point Tests

By generating several GARCH(1,1) data sets with varying parameters then gluing these data sets together, we can effectively create a GARCH(1,1) series that has a specified number of change points. We would also know the exact location of these change-points and the parameters of the modeled segments. This will provide us with an excellent source for verifying that our testing methods will detect and pick out these generated change-points.

The first change-point generated data that we consider consists of four different segments, each of size 300, with the following parameters.

(i) $\omega = 1$, $\alpha = 0$, $\beta = 0$

(ii) $\omega = 1$, $\alpha = 0.5$, $\beta = 0.4$

(iii) $\omega = 1$, $\alpha = 0$, $\beta = 0$

(iv) $\omega = 1$, $\alpha = 0.5$, $\beta = 0.4$
| \( \hat{B}_n \) Statistic on British Pound Data Beginning with Entire Data Set |
|-----------------------------|------------------|
| **Week**                   | 403              |
| **\( \hat{B}_n \) Value**  | 25.92681         |
| **p-value**                | \( p < 0.01 \)   |
| **Week**                   | Not Sig.         |
| **\( \hat{B}_n \) Value**  | **2.755199**     |
| **p-value**                | 0.0527 < \( p \) < 0.0560 |
|                            | Not Sig.         |
|                            | 1.403558         |
|                            | 0.3146 < \( p \) < 0.3386 |

<table>
<thead>
<tr>
<th><strong>Final Segments and Corresponding GARCH(1,1) Parameter Estimates</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Weeks</strong></td>
</tr>
<tr>
<td><strong>( \omega )</strong></td>
</tr>
<tr>
<td><strong>( \alpha )</strong></td>
</tr>
<tr>
<td><strong>( \beta )</strong></td>
</tr>
</tbody>
</table>

Table 11: Segmentation Summary of British Pound Data Using \( \hat{B}_n \) Statistic

The changes from one segment to the other produce three definite change-points at 300, 600, and 900 as seen by the following time-series plot of Figure 5.

Figure 5: Time Series Plot of Generated Data with Change Points at 300, 600 and 900
Surprisingly the $\hat{M}_n$ and $\hat{B}_n$ change-point test statistics both result in non-significant values. The exact $\hat{M}_n$ value is found to be 0.7124021 which corresponds to a $p$-value of approximately 0.10. Similarly, the $\hat{B}_n$ statistics yields a value of 1.853846 which corresponds to a $p$-value of about 0.18. The following Figure 6 shows the plots of the statistics to both be hinting at a change-point around 300, but again, these tests were not statistically significant in detecting the change.

![Figure 6: $\hat{M}_n$ and $\hat{B}_n$ Values From Generated Data with Change Points at 300, 600, and 900](image)

Assuming this was a special case, and the tests really are accurate or powerful in detecting changes, we create another data set with the same size and parameters as the previous. Again our results are not significant in detecting the change-points. The resulting $\hat{M}_n$ and $\hat{B}_n$ values are 0.749689 and 2.067421 with $p$-values of approximately 0.065 and 0.14 respectively. This is an indication that our tests are possibly not very powerful. However, this is only an indication and does not necessarily prove the notion. In order to assert this claim, one would have to generate the
same change point model many times and see what the percentage of rejections (or detections) is. This would give the empirical power of the tests. This paper does not engage in such an exercise. It is also possible that there are changes other than changes in the model parameters that are responsible for our highly significant rejections found in the German Mark and British Pound Data. Some possible explanations include that maybe the whole model is changing, or maybe the innovations are not normal.

6 Summary and Conclusions

In our efforts to detect change-points, the test statistic $\Lambda^*_n$ was derived from the familiar Generalized Likelihood Ratio. This was our test of choice and would have provided the best power in terms of detecting change-points. Unfortunately, the distribution of $\Lambda^*_n$ was determined to depend on the underlying model, and therefore one would have to rely on bootstrap or Monte Carlo methods in order to accurately estimate the empirical quantiles for the particular model and data.

Due to this dependence we considered two other change-point test statistics that were both based on the sequential empirical process. The first of the two being the $M_n$ Kolmogorov-Smirnov type statistic, and the second being the $B_n$ Cramer-von Mises type statistic. These statistics were then used to detect possible change-points in several years of foreign currency data on the German Mark and the British Pound.

The results from performing these tests on the two data sets indicate that the $M_n$ statistic might be slightly more sensitive in detecting change-points than is the $B_n$ statistic. Both tests, however, indicate that there is a change around week 400. This corresponds to 1979 when the European Monetary System was introduced, which established limits or bounds on how much a currency rate could change in a year's time. This fact provides good supporting evidence that indeed a change-point could exist at that time in the data sets. The tests were also performed on a pair of simulated data sets that included change-points. The results from this analysis suggest that both the $M_n$ and the $B_n$ tests may not be very powerful. Again, these results do not necessarily prove the weakness of the tests, but only indicate or suggest that they may not be optimal. A more thorough simulation study consisting in generating the same data and performing the change-point tests repeatedly would give the accurate empirical power of the above-mentioned tests.
Appendix

A Code used in formulating Table 1 output:

```r
bias.mse_function(n, a.value, arch, garch,R){
    abias_rep(0,R)
alphabias_rep(0,R)
betabias_rep(0,R)

    for( i in 1:R){
        g.mod_list(a.value=a.value, arch=arch, garch=garch)
y_simulate.garch(g.mod, n=n, n.start=100 )$et

        if(garch==0)est.garch(formula.mean= ~-1, formula.var= ~ garch(1,0), series = y, trace=F)
    else est.garch(formula.mean= ~-1, formula.var= ~ garch(1,1), series = y, trace=F)

        theta_est$coeff[,1]
        abias[i]_(g.mod$a.value-theta[1])
alphabias[i]_(g.mod$arch-theta[2])
betabias[i]_(g.mod$garch-theta[3])
        print(i)
    }

totabias_sum(abias)/R
totalalphabias_sum(alphabias)/R
totbetabias_sum(betabias)/R

    amse_sum(abias^2)/R
    alphamse_sum(alphabias^2)/R
    betamse_sum(betabias^2)/R

    return(totabias, totalalphabias, totbetabias, amse, alphamse, betamse)
}

biasmse.500.2.5.0.1000.bias.mse(500,.2,.5,0,1000)
```

32
B Code used in formulating Table 2 output:

```
B Code used in formulating Table 2 output:

n_100   a.value_1   arch_4   garch_0
lag_10   R_1000   mlr_rep(0,R)   lr_rep(0,n)   a_rep(0,n)

glr_function(){
  For(r= 1:R, {
    module(garch)
    g.mod_list(a.value=a.value, arch=arch, garch=garch)
    y_simulate.garch(g.mod, n=n, n.start=100)
    estn_garch(formula.mean=~1, formula.var= ~ garch(1,0), series = y$et, trace=F)
    for(t in 1:n ) { a[t]._(y$et[t]/estn$sigma.t[t])^2+log((estn$sigma.t[t])^2)}
    for(k in (lag+1):(n-lag)){
      b_rep(0,k)
      c_rep(0,n)

      new1_y$et[1:k]
      estk_garch(formula.mean=~1, formula.var= ~ garch(1,0), series = new1 , trace=F)
      for(t in 1:k ) { b[t]._(y$et[t]/estk$sigma.t[t])^2+log((estk$sigma.t[t])^2)}

      new2_y$et[(k+1):n]
      estkn_garch(formula.mean=~1, formula.var= ~ garch(1,0), series = new2 , trace=F)
      for(t in (k+1):n){ c[t]._(y$et[t]/estkn$sigma.t[t-k])^2+log((estkn$sigma.t[t-k])^2)}

      lr[k]._(sum(a)-sum(b)-sum(c))
    }
    mlr[r].max(lr[(lag+1):(n-lag)])
    print("r")
    print(r)
    print("mlr")
    print(mlr)
  }, grain.size=15)
  list(mlr=mlr)
}
glr.100.1.4.0.10.1000_glr() #n=100 a=.1 arch=.4 garch=0 lag=10 iterations=1000
mlr=glr.100.1.4.0.10.1000$mlr
```

33
C Code used in formulating Table 4 output:

Msup_rep(0,100) finiteT_100 MTn_rep(0,finiteT)

table.test.mn_function() {
For(z 1:100, {
  normsq_(rnorm(n=100,mean=0,sd=1))^2
  tmax_(max(normsq))
  print(z)
  n_length(normsq)
  F1k_rep(0,n)
  Fkn_rep(0,n)
  Tn_rep(0,n)
  MTn_rep(0,finiteT)
  for(j in 1:finiteT) {
    value_((j*tmax)/finiteT)
    for(k in 1:(n-1)) {
      counta_0
      countb_0
      temp1k_matrix(normsq[1:k],ncol=1)
      counta_sum(ifelse(temp1k < value,1,0))
      tempkn_matrix(normsq[(k+1):n],ncol=1)
      countb_sum(ifelse(tempkn < value,1,0))
      F1k[k]-(1/k)*counta
      Fkn[k]-(1/(n-k))*countb
      Tn[k]_sqrt(n)*(k/n)*(1-(k/n))*abs(F1k[k]-Fkn[k])
    }
    MTn[j]_max(Tn)
  }
  Msup[z]_max(MTn)
}, grain.size=5) list(Msup=Msup, MTn=MTn)
}
test.table100.sigma.norm_table.test.mn() test.table2.sigma.norm$MTn

34
D Code used in formulating Tables 6 and 7 output:

\[
\text{zsum}\_\text{rep}(0,100) \quad \text{small}\_1/10000
\]

table.test.blum.mn\_function() {
    For(z 1:100, {
        normsq\_\text{rnorm(n=100,mean=0,sd=1)}^2
        n\_\text{length(normsq)}
        F1k\_\text{rep(0,n)}
        Fkn\_\text{rep(0,n)}
        Tn\_\text{rep(0,n)}
        jsum\_\text{rep(0,n)}

        for(j in 1:n) {
            for(k in 1:(n-1)) {
                counta\_0
                countb\_0

                temp1k\_\text{matrix(normsq[1:k],ncol=1)}
                counta\_\text{sum(ifelse(temp1k < (normsq[j]+small),1,0))}

                tempkn\_\text{matrix(normsq[(k+1):n],ncol=1)}
                countb\_\text{sum(ifelse(tempkn < (normsq[j]+small),1,0))}

                F1k[k]\_\text{(1/k)}*counta
                Fkn[k]\_\text{(1/(n-k))} *countb

                Tn[k]\_\text{(sqrt(n)*(k/n)*(1-(k/n)))*abs(F1k[k]-Fkn[k]))^2}
            }
            jsum[j]\_\text{sum(Tn)}
        }
        zsum[z]\_\text{5*(3.14159265359)^4*sum(jsum)/(n^2)}
        print(z)
    }, grain.size=5)
    list(zsum=zsum)
}

test.table100.sigma.norm.blum\_table.test.blum.mn() \quad \text{test.table2.sigma.norm.blum$zsum}
E  Code used in formulating values for Figure 2

module(garch) y_week.lrusdm.dat
estn_garch(formula.mean= ~1, formula.var= ~ garch(1, 1), series = y, trace=F)
resq_(y/estn$sigma.t)^2 n_length(y)
F1k.0 Fkn.0 Tn_rep(0,n)
MTn_rep(0,n) small.1/10000 resqsm_resq+small

mnFdmwk_function(){
  For(k = 1:(n-1),{
    temprlk_matrix(resq[1:k],ncol=1)
temprkn_matrix(resq[(k+1):n],ncol=1)
    for(j in 1:n){
      cta.0 ctb.0
      F1k.0 Fkn.0
      cta_sum(ifelse(temprlk < resqsmLi],1,0))
tcb_sum(ifelse(temprkn < resqsmLi],1,0))
      F1k_.(1/k)*cta
      Fkn_.(1/(n-k))*ctb
      Tn[j].sqrt(n)*(k/n)*(1-(k/n))*abs(F1k-Fkn)
    } MTn[k].max(Tn)
    print(k)
  }, grain.size=30)
  Msup_max(MTn)
  list(Msup=Msup, MTn=MTn)
}
mnFdmwk.second_mnFdmwk() mnFdmwk.second$MTn
mnFdmwk.second$Msup #returns week 443 (2.076593) !!!!
module(garch) y_week.lrusdm.dat n.length(y)
small.1/100000 F1k.0 Fkn.0
Tn__rep(O,n) ksum__rep(O,n)
estn.garch(formula.mean= ~ -1, formula.var= ~ garch(1,1), series = y, trace=F)
resq_(y/estn$sigma.t) A 2 resqsm__resq +small

Bdm1.function(){
  For(k = 1:(n-1),{
    temp1lk matrix(resq[1:k],ncol=1)
    tempkkn matrix(resq[(k+ 1):n],ncol=1)
    for(j in 1:n){
      cta_0 ctb_0
      F1k_0 Fkn_0
      cta.sum(ifelse(temp1lk < resqsm[j],1,0))
      ctb.sum(ifelse(tempkkn < resqsm[j],1,0))
      F1k_.(1/k)*cta
      Fkn_.(1/(n-k))*ctb
      Tn[j].(sqrt(n)*(k/n)*(1-(k/n))*abs(F1k-Fkn))^2
    }
    ksum[k].sum(Tn)
    print(k)
  }, grain.size=30)
  fsum_.5*(3.14159265359)^4*sum(ksum)/(n^2)
  list(fsum=fsum, ksum=ksum)
}
FBdm1wk.Bdm1() FBdm1wk$ksum FBdm1wk$fsum

37
G++ Code provided by Gudrun Kokoszka (GLRstat.h)

class GLRSTAT:pubic ARCH{

public:

GLRSTAT(int *ipar, double *dpar, long int &seed, char *f1);
~GLRSTAT();

void GLRstart(double *dparam);
void GLRsimulation();
void GLRestimation();
void CopyCoefs();
void GLR_ARCH_estimation();
void AssignY(int s, int e);
void CoefsTilda();
void CoefsBar();

private:

double * bhat;
double * bbar;
double * btilda;
double * Ysave;
double result;

int k;
in n;
}


#include "arch.h"
#include "GLRstat.h"

extern "C" {
    char *strcat (char *dest, const char *src);
    double log (double s);
}

extern void nrerror(char*);
extern void dfpmin(double *p, ARCH & msc, double(*func)(double *,ARCH &),
void(*dfunc) (double*,double*,double(*f) (double*,ARCH &),ARCH&,double));
extern void num_derv (double *x, double *gr, double(*f)(double*,ARCH &),
ARCH & msc, double fx);

GLRSTAT::GLRSTAT() {
    delete [] bhat;
    delete [] btilda;
    delete [] bbar;
    delete [] Ysave;
}

GLRSTAT::GLRSTAT(int *ipar, double *dpar, long int &seed, char *fl)
:bhat(0),bbar(0),btilda(0),Ysave(0){
    nobs = ipar[0];
    p = ipar[1];
    DGP = ipar[2];
    PSV = ipar[3];
    NBR = ipar[4];
    TYPE = ipar[5];
    k = ipar[6];

    idum = seed;
    idum1 = -1000+idum;
file_out1 = new char [14];
strcpy(file_out1,f1);

n = nobs;
int nrows = n+1-2*k;

GLRstart(dpar);
}

void GLRSTAT::GLRstart(double * dparam){

    int i;
    char * info_file;

    if (DGP == 1 && TYPE == 1) NVAR = 3;
    if (DGP == 2 && TYPE == 1) NVAR = p+1;
    if (DGP == 1 && TYPE == 2) NVAR = 4;
    if (DGP == 2 && TYPE == 2) NVAR = p+2;

    Y = new double [nobs+1];
    bstar = new double [NVAR+1];
    b0 = new double [NVAR];
    bhat = new double[NVAR+1];
    btilda = new double[NVAR+1];
    bbar = new double[NVAR+1];
    Ysave = new double[nobs+1];

    for (i=0;i < NVAR;i++)
        b0[i] = dparam[i];

    info_file = new char [40]; // information file
    strcpy(info_file,file_out1);
    strcat(info_file," .info "); // concatenate the suffix info
    ofstream ecrire0(info_file,ios::out);
    ecrire0 <<"--- --------- - -"<<endl;
    ecrire0 <<" Output file: "<file_out1<<endl;
if (TYPE == 1) ecrire0 <<" Number of simulations: " <<NBR <<endl;
ecrire0 <<" Type of Model: ";
if (DGP == 1) ecrire0 <<"GARCH(1,1) and GLR statistic" <<endl;
if (DGP == 2) ecrire0 <<"ARCH(" <<p <<") and GLR statistic" <<endl;
if (TYPE == 1) ecrire0 <<" SIMULATION" <<endl;
if (TYPE == 1)
{
ecrire0 <<" PARAMETERS FOR SIMULATION: ";
for (i=0;i < NVAR;i++)
ecrire0 <<b0[i]<<" ";
ecrire0 <<endl;
}

ecrire0 <<"--------------------------------------------------------" <<endl;
ecrire0.close();

GLRsimulation();

}

void GLRSTAT::GLRsimulation()
{
    int i,j;
    ofstream ecrirel(file_out1,ios::out);
    OK=1;

    for (i =1; i<=NBR;i++)
if (OK == 1)
cout <<"Replication : " <<i <<endl;

ARCH_DGP(idum,idum1);
ARCH_estimation();

for (int m=1;m<=nobs;m++)
Ysave[m] = Y[m];
if (OK == 0) i--;
else
{
    // save bstar coefs in bhat

    for (int l=0;l<=NVAR;l++)
    btilda[l]=0;
    bbar[l]=0;
    bhat[l] = bstar[l+1];
}
GLRestimation();

ecrirel << i << ". ";

// for (j=0;j<NVAR;j++)
// ecrirel << " b[" << j << "] " << bhatLi] << " ";

ecrirel << result << endl;
}

} 

ecrirel << "End of file " << endl;
ecrirel.close();

}

void GLRSTAT::GLRestimation(){

double shat=0;
double stilda=0;
double sbar=0;

double rhat=0;
double rtilda=0;
double rbar=0;

double temp;
double t;

// compute result rhat once

for (int p=2; p<= n; p++){
    shat = bhat[0] + bhat[1] * (Y[p-1]*Y[p-1]);
    t = log(shat);
    rhat += (Y[p]*Y[p])/shat + t;
}

// start estimation of other coefficients

int times = n-2*k;
result = -9999999;

for (int i = 1; i <= times; i++){

    // results of sub-samples

    rtilda=0;
    rbar=0;

    // btilda first

    Assign Y(1,k+i-1);
    nob = k+i-1;
    GLR_ARCH_estimation();
    CoefSTilda();

    for (int l=2; l<=(k-1)+i; l++){
        stilda = btilda[0] + btilda[1]*(Y[l-1]*Y[l-1]);
        t = log(stilda);
        rtilda += (Y[l]*Y[l])/stilda + t;
    }
void GLRSTAT::GLR_ARCH_estimation(void)
{
    // any check on the estimates of the sub-samples has been taken out!
    int i;
    double *x;

    roundoff = 0;
    pos_def = 1;
    //OK = 0;
    x = new double[NVAR+1];

    for (i=1; i <= NVAR; i++) // WARNING x STARTS AT ONE
    x[i] = b0[i-1];

    double (*likelihood)(double*, ARCH&);
    if (DGP == 1)
        likelihood = lik_garchll;
    else if (DGP == 2)
        likelihood = lik_garch11;
    else if (DGP == 2)
likelihood = lik_archp;

dfpmin(x,*this,likelihood,num_derv);

//if (roundoff == 0)
//{
hessian(x,likelihood);
//if (pos.def == 1) OK = 1;
//test.estimates(x);
//}

//if (OK == 1)
for (i=1;i <= NVAR;i++)
bstar[i] = x[i]; // SAVE ESTIMATION RESULTS IN GLOBAL b.

delete []x;
}

void GLRSTAT::AssignY(int s, int e){
    delete [] Y;
    int size = e-s+2;
    int t;
    Y = new double[size];
    for (int i=s; i<=e; i++) t=i-s+1; Y[t] = Ysave[i]; }
}

void GLRSTAT::CoefsTilda(){
    for (int i=0; i<=NVAR;i++) btilda[i] = bstar[i+1];
}

void GLRSTAT::CoefsBar(){
    for (int i=0; i<=NVAR;i++) bbar[i] = bstar[i+1];
}
I C++ Code provided by Gilles Teyssièr (arch.h)

#include<iostream.h>
#include<fstream.h> // Gu took off .h extension
using std::cout; // Gu had to specify identifiers
using std::cerr;
using std::endl;
using std::ofstream;
using std::ios;
using std::istream;
using std::ifstream;
using std::cin;
#endif

class ARCH

{ public:

    double *Y, *b0, *bstar;

    int DGP, PSV, NBR, NVAR, TYPE, roundoff, pos_def, OK;
    int nobs, p;
    long int idum, iduml;
    char *file_outl, *fileJn;

    void ARCH_DGP(long int &idum, long int &iduml);
    void ARCH_estimation(void);
    void simulation(void);
    void estimation_on_real_data(void);
    void start(double *param);
    void quit(void);
    void OLS (double *beta_hat, int nvar);
    void tests_estimates(double *x);
void hessian(double *x, double(*func)(double*,ARCH &));
double normal(double mean, double variance, long &idum, long &idum1);

double gtol;
int nvar(void) {return (NVAR);} 
int read_nb_obs(void) {return nabs;}
void cv(int i) {roundoff = i;}

// CONSTRUCTOR FOR SIMULATION /Gu initialized pointers to NULL
ARCH(int *ipar, double *dpar, long int &seed,char *fl):Y(0),b0(0),bstar(0),file_outl(0)
{

nobs = ipar[0];
p = ipar[1];
DGP = ipar[2];
PSV = ipar[3];
NBR = ipar[4];
TYPE = ipar[5];

idum = seed;
idum1 = -1000+idum;

file_outl = new char [14];
strcpy(file_outl,fl);
start(dpar);
}

// CONSTRUCTOR FOR ESTIMATION ON REAL DATA
ARCH(int *ipar, double *dpar, char *fl, char *f2)
{
nobs = ipar[0];
p = ipar[1];
DGP = ipar[2];
PSV = ipar[3];
TYPE = ipar[5];
file_out1 = new char [20];
strcpy(file_out1,f1);
file_in = new char [20];
strcpy(file_in,f2);

start(dpar);
}

// Gu NEW default CONSTRUCTOR

ARCH(): Y(0), b0(0), bstar(0), file_out1(0)
{
nobs = 0;
p = 0;
DGP = 0;
PSV = 0;
TYPE = 0;

file_out1 = new char [20];
strcpy(file_out1,"test");
file_in = new char [20];
strcpy(file_in,"check");

}

ARCH(void){quit();} //DESTRUCTOR

friend double lik_archp(double *x, ARCH & boot);
friend double lik_garch11(double *x, ARCH & boot);
friend double lik_archp_constant(double *x, ARCH & boot);
friend double lik_garch11_constant(double *x, ARCH & boot);
};

const double EPSMCH(2.22046e-15);

#define G_METHOD 1
#define HCCM 1

48
const double SCALE(-1.0);

/////////// END OF FILE ARCH.h /////////////
C++ Code provided by Gilles Teyssière (arch-main.ccp)

#include "arch.h"

extern "C"
{
    void exit(int code);
    double floor(double);
    char *strcat (char *dest, const char *src);
    int atoi (const char *s);
}

template <class Ti>
T ** matrix (T **, int nrl, int nrh, int ncl, int nch)
{
    T **m;
    if (!(m = new T *[nrh+1]))
        nrerror(" ALLOCATION FAILURE 1 IN MATRIX");
    for (int i=1; i<= nrh; i++)
        if (!(m[i] = new T [nch+1]))
            nrerror(" ALLOCATION FAILURE 2 IN MATRIX");
    return m;
}

template <class Ti>
void free_matrix(T **m, int nrl, int nrh, int ncl)
{
    for (int i=nrh; i>= 1; i--)
        delete [] m[i];
    delete [] m;
}
extern void dfpmin(double *p, ARCH& msc, double(*func)(double *,ARCH &),
    void(*dfunc)(double*,double*,double(*f)(double*,ARCH &),ARCH&,double));
extern void num_derv (double *x, double *gr, double(*f)(double* ,ARCH & ) ,
    ARCH & msc, double fx);
extern void chol_inv(double **a, int n, int &pd);

void nrerror(char *error_text)
{
    cerr <<"Erreur dans une routine numerique...
    cerr <<error_text;
    cerr <<"\n .... Retour au systeme...
    exit(1);
}

void ARCH::test_estimates(double *x)
{
    int i;
    double sum(0.0);
    OK = 1;

    for (i=1; i<= p+1;i++)
        if (x[i]<= 1.0e-12) OK = 0;

    for (i=2; i<= p;i++)
        sum += x[i];
    if (sum > 1.0) OK = 0;

    if (DGP == 1)
    {
        double a,b;
        a = x[2];
        b = x[3];
        if (a*a + 2*a*b+ 3*b*b >= 1.0) OK = 0;
    }
}

void ARCH::OLS(double *beta_hat, int nvar)
```c
{  
  int size(nobs-nvar), i, j, k, pd(0);
  double *ty, *xty, **tx, **xtx, *SY;

  SY = new double[nobs+1];
  ty = new double[size+1];
  xty = new double[nvar+1];
  tx = matrix(tx,1,size,1,nvar);
  xtx = matrix(xtx,1,nvar,1,nvar);

  for (i=l;i <= nobs;i++)
  SY[i] = Y[i]*Y[i];

  for (i=l;i <= size;i++)
  {
    ty[i] = SY[i+nvar];
    tx[i][1] = 1.0;
    for (j=l;j < nvar;j++)
      *(*(tx+i)+j+l)= SY[i+nvar-j];
  }

  for (i=l;i <= nvar; i++)
  {
    xty[i] = beta_hat[i] = 0.0;
    for (j=i;j <= nvar;j++)
      *(*(xtx+i)+j) = *(xtx+j+i) = 0.0;
  }

  for (i=l;i <= nvar; i++)
  for (j=l;j <= nvar;j++)
  for (k=l;k <= size;k++)
    *(xtx+i)+j) += (*(tx+k)+i) * (*(tx+k)+j));

  chol_inv(xtx,nvar,pd);
  for (i=l;i <= nvar;i++)
  for (j=l;j <= size;j++)
    xty[i] += (*(tx+j)+i)*ty[j];
```
for (i=1;i <= nvar;i++)
    for (j=1;j <= nvar;j++)
        beta_hat[i] += (*(*(xtx+i)+j))*xtyLl;

delete [] SY; delete [] ty; delete [] xty;
free_matrix(tx,1,size,1);
free_matrix(xtx,1,nvar,1);
}

void ARCH::ARCH_estimation(void)
{
    int i;
    double *x;

    roundoff = 0;
pos_def = 1;OK = 0;
x = new double[NVAR+1];

    for (i=1;i <= NVAR;i++) // WARNING x STARTS AT ONE
        x[i] = b0[i-1];

    double ((*likelihood)(double*, ARCH&));
    if (DGP == 1)
        likelihood = lik_garch11;
    else if (DGP == 2)
        likelihood = lik_archp;

dfpmin(x,*this,likelihood,num_derv);

    if (roundoff == 0)
    {
        hessian(x,likelihood);
        if (pos_def == 1) OK = 1;
test_estimates(x);
void ARCH::simulation(void)
{
    int i, j;

    ofstream ecrire1(file_out1, ios::out);

    OK = 1;
    for (i = 1; i <= NBR; i++)
    {
        if (OK == 1)
            cout << "Replication : " << i << endl;
        ARCH_DGP(idum, idum1);
        ARCH_estimation();
        if (OK == 0) i--;
        else
        {
            for (j = 1; j <= NVAR; j++)
                ecrire1 << bstar[j] << " ";
            ecrire1 << endl;
        }
    }

    ecrire1.close();
}

void ARCH::estimation_on_real_data(void)
{
    double *x, mu;

    if (OK == 1)
        for (i = 1; i <= NVAR; i++)
            bstar[i] = x[i]; // SAVE ESTIMATION RESULTS IN GLOBAL b.

        delete []x;
    }
int i;
long double sum(0.0);

ifstream lire(file_in, ios::in);
for (i=1; i <= nobs;i++)
{
    lire >> Y[i];
    Y[i] *= 100.0;
}
lire.close();

ofstream ecrire1(file_out1, ios::out);

if (DGP == 2) // ARCH (p) MODEL
{
    double *beta_hat, *tempY;
	x = new double[p+3];
tempY = new double[nobs+1];

for (i=1;i <= nobs;i++)
{
    sum += Y[i];
    tempY[i] = Y[i];
}

mu = double(sum)/nobs;
for (i=1;i <= nobs;i++)
    Y[i] = Y[i]-mu;

beta_hat = new double[p+2];

OLS(beta_hat,p+1);
for (i=1; i <= nobs; i++)
    Y[i] = tempY[i];

for (i=1; i <= p+1; i++)
{
    x[i] = beta_hat[i];
    if (x[i] <= 1.0e-8) x[i] = b0[i-1];
}

x[p+2] = mu;

delete [] beta_hat;
delete [] tempY;

double ((*likelihood)(double*, ARCH&));
likelihood = lik_archp_constant;
dfpmin(x,*this,likelihood,num_derv);

if (roundoff == 0)
{
    hessian(x,likelihood);
    if (pos_def == 1) OK = 1;
    test_estimates(x);
}
if (OK == 1)
{
    for (i=1; i <= NVAR; i++)
        ecrirel << x[i] << " ";
    ecrirel << endl;
}
delete [] x;

}

else if (DGP == 1) // GARCH (1,1) MODEL
{
    for (i=1; i <= nobs; i++)
sum += Y[i];

mu = double(sum)/nobs;

sum = 0.0;
for (i=1;i <= nobs;i++)
    sum += (Y[i]-mu)*(Y[i]-mu);

x = new double[5];
x[1] = (double(sum)/(nobs-1.0))/10.0;
x[2] = 0.70;
x[3] = 0.10;
x[4] = mu;

double (*likelihood)(double*, ARCH&));
likelihood = lik_garch11_constant;
dfpmin(x,*this,likelihood,num_deriv);

if (roundoff == 0)
{
    hessian(x,likelihood);
    if (pos_def == 1) OK = 1;
    test_estimates(x);
}
//if (OK == 1)
{
    for (i=1;i <= NVAR;i++)
        ecrire1 <<x[i]" ";
    ecrire1<<endl;
}
delete [] x;
}

ecrire1.close();
}
void ARCH::start(double *dparam)
{
    int i;
    char * info_file;

    if (DGP == 1 && TYPE == 1) NVAR = 3;
    if (DGP == 2 && TYPE == 1) NVAR = p + 1;
    if (DGP == 1 && TYPE == 2) NVAR = 4;
    if (DGP == 2 && TYPE == 2) NVAR = p + 2;

    Y = new double [nobs + 1];
    bstar = new double [NVAR + 1];
    b0 = new double [NVAR];

    for (i = 0; i < NVAR; i++)
        b0[i] = dparam[i];

    info_file = new char [40]; // information file
    strcpy(info_file, file_out1);
    strcat(info_file, " .info"); // concatenate the suffix info
    ofstream ecrire0(info_file, ios::out);
    ecrire0 << "-------- -- --- -" << endl;
    ecrire0 << " Output file: " << file_out1 << endl;
    if (TYPE == 1) ecrire0 << " Number of simulations: " << NBR << endl;
    ecrire0 << " Type of Model: ";
    if (DGP == 1) ecrire0 << " GARCH(1,1)" << endl;
    if (DGP == 2) ecrire0 << " ARCH(" << p << ")" << endl;
    if (TYPE == 1) ecrire0 << " SIMULATION" << endl;
    if (TYPE == 1)
    {
        ecrire0 << " PARAMETERS FOR SIMULATION: ";
        for (i = 0; i < NVAR; i++)
            ecrire0 << b0[i] << " ";
        ecrire0 << endl;
    }

    if (TYPE == 2) ecrire0 << " ESTIMATION ON REAL DATA" << endl;

58
if (TYPE == 2) ecrire0 << " INPUT FILE: " << file_in << endl;

ecrire0 << "---------------------------------------------." << endl;

ecrire0.close();

if (TYPE == 1)
simulation();
else if (TYPE == 2)
estimation_on_real_data();
}

void ARCH::quit(void)
{

delete [] Y;
delete [] b0;
delete [] bstar;
// delete [] file_out1;
}

#include "GLRstat.h" // new derived class specification for GLR statistic comp.

int main()
{

    char * arg1 = new char[30]; // Gu in DOS start *.exe 123 100 disregards passed arguments
    - maybe my setup
    char * arg2 = new char[30]; // chose to ask for user input instead

    cout << " Please enter the seed: " ;
cin >> arg1;
    cout << " Please enter number of replications: ";
cin >> arg2;

    // Rest of the code
long int seed(-atoi(arg1));
int nb_replic(atoi(arg2));

delete [] arg1;
delete [] arg2;

double *dparam;
int *iparam;
char *file_out, *file_in;

iparam = new int[8]; // Gu added another parm for GL estimation - now 7 not 6
dparam = new double[3]; // should be equal to the number of estimated parameters.
file_out = new char[20];

strcpy(file_out,"glrarch"); // name of the result file
strcat(file_out,"l"); // concatenate the number used for the seed
strcat(file_out,".dat"); // concatenate the suffix dat

// iparam[0] = 30; //number of observations

cout << endlNumber of observations: ";
cin >> iparam[0];

iparam[1] = 1; //order of the ARCH process. Should be set to 2 for GARCH(1,1)
iparam[2] = 2; //type of DGP 1 for GARCH(1,1) 2 for ARCH(p)
iparam[3] = 10; //</size of pre-sample values
iparam[4] = nb_replic; //number of replications
iparam[5] = 1; // type of program: 1 for simulation, 2 for estimation.
cout << endlMinimum k value: ";
do {
  cin >> iparam[6]; // minimum k value
  if (iparam[6]*2 >= iparam[0]) cout << endlPlease enter a smaller k value: ";
} while (iparam[6] * 2 ;= iparam[0]);

iparam[7] = 1;
cout << endlnb[0] value: ";
cin >> dparam[0];
cout << "\nb[1] value: ";
cin >> dparam[1];
dparam[2] = 0.20; // b2 DGP

if (iparam[5] == 1) { // SIMULATION
    if (iparam[7] == 1) // Gu new GLRstat constructor call
        GLRSTAT rich(iparam,dparam,seed,file_out);
    else
        ARCH rich(iparam,dparam,seed,file_out);
else // ESTIMATION FROM REAL DATA
{
    file_in = new char[20];
    strcpy(file_in,"lrusdm.dat");
    ARCH rich(iparam,dparam,file_out,file_in);
    delete [] file_in;
}

delete [] dparam;
delete [] iparam;
delete [] file_out;

    return 0;
}

/////////// END OF FILE ARCH-main.cpp /////////////
K  C++ Code provided by Gilles Teyssièere (arch-dgp.cc)

// ARCH-DGP
// GILLES. 11 FEBRUARY 2001

#include "arch.h"
#include "GLRstat.h"

extern "C" {
  double sqrt(double);
  double log(double);
}

const double PI(3.141592653589793);
const double LN2PI(log(2.0*PI));

extern double gasdev(long int &idum,long int &iduml);
extern int inLrand(int n,long &idum, long &iduml);
extern void sort (unsigned long n, double *arr);

double ARCH::normal(double mean, double variance, long int &idum, long int &iduml)
{
  return (sqrt(variance)*gasdev(idum,iduml)+mean);
}

void ARCH::ARCH_DGP(long int &idum, long int &iduml)
{
  double *y;
  long double sigma;
  double sigma0;
  int i, j;

  y = new double [nobs+PSV+1];
if (DGP == 1) // GARCH(1,1) DGP
{
    sigma0 = b0[0];
    for (i=1; i <= nobs+PSV;i++)
    {
        sigma = b0[0] + b0[1]*sigma0 + b0[2]*y[i]*y[i-1];
        sigma0 = double(sigma);
        y[i] = sqrt(sigma0)*normal(0,1,idum,idum1);
    }
}

else if (DGP == 2) // ARCH (p) DGP
for (i=1;i <= nobs+PSV;i++)
{
    sigma = b0[0];
    for (j=1;j<=p && j<i;j++)
    
        sigma += b0[j]*(y[i-j]*y[i-j]);
    y[i] = sqrt(double(sigma))*normal(0,1,idum,idum1);
}

for (i=1;i <= nobs;i++)
    Y[i] = y[i+PSV];

delete[] y;

/////////////////////////LIKELIHOOD FUNCTION OF AN ARCH(p)//////////////////////////////////////////

double lik_archp(double *x, ARCH & boot)
{
    double sigma, omega, *u, *b;
    long double lik(0.0);
    int i, j, nobs(boot.nobs), p(boot.p);

    for (i=1;i <= p+1;i++)
if (x[i] < 1.0e-13) x[i] = 1.0e-13;

omega = x[1];
b = x[1];

u = new double[nobs+1];

for (i=1;i <= nobs;i++)
{
    u[i] = boot.Y[i]*boot.Y[i];
    sigma = omega;
    for (j=1;j<=p && j<i;j++)
        sigma += b[j]*u[i-j];
    if (sigma < 1.0e-12) sigma = 1.0e-12;
    lik += log(sigma)+u[i]/sigma;
}

delete [] u;
return (-0.5*(nobs*LN2PI + double(lik))*SCALE);

///////////////////////////////////////////////////////
// LIKELIHOOD FUNCTION OF A GARCH(1,1)
///////////////////////////////////////////////////////

double lik_garch11(double *x, ARCH & boot)
{
    double sigma, sigma0, omega, *u, alpha, beta;
    long double lik(0.0);
    int i, j, nobs(boot.nobs);

    if (x[1] < 1.0e-13) x[1] = 1.0e-13;

    omega = x[1];
beta = x[2];
alpha = x[3];

u = new double[nobs+1];

sigma0 = omega;
u[0] = 0.0;
for (i=1;i <= nobs;i++)
{
    u[i] = boot.Y[i]*boot.Y[i];
    sigma = omega + beta*sigma0 + alpha*u[i-1];
    if (sigma < 1.0e-12) sigma = 1.0e-12;
    sigma0 = sigma;
    lik += log(sigma)+u[i]/sigma;
}

delete [] u;
return (-0.5*(nobs*LN2PI + double(lik))*SCALE);
}

///////////////////////////////////////////////////////////////////////////////
// LIKELIHOOD FUNCTION OF AN ARCH(p) WITH CONSTANT
///////////////////////////////////////////////////////////////////////////////
double lik_archp_constant(double *x, ARCH & boot)
{
    double sigma, omega, mu, temp, *u, *b;
    long double lik(0.0);
    int i, j, nobs(boot.nobs), p(boot.p);

    for (i=1;i <= p+1;i++)
        if (x[i] < 1.0e-13) x[i] = 1.0e-13;

    omega = x[1];
    mu = x[2];
    b = x+2;

    u = new double[nobs+1];
for (i=1;i <= nobs;i++)
{
    temp = boot.Y[i]-mu;
    u[i] = temp*temp;
    sigma = omega;
    for (j=1;j<=p & & j<i;j++)
        sigma += b[j]*u[i-j];
    if (sigma < 1.0e-12) sigma = 1.0e-12;
    lik += log(sigma)+u[i]/sigma;
}

delete [] u;
return (-0.5*(nobs*LN2PI+ double(lik))*SCALE);
}

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\}
for (i=1;i <= nobs;i++)
{
    temp = boot.Y[i]-mu;
    u[i] = temp*temp;
    sigma = omega+ beta*sigma0 + alpha*u[i-1];
    if (sigma < 1.0e-12) sigma = 1.0e-12;
    sigma0 = sigma;
    lik += log(sigma)+u[i]/sigma;
}

delete [] u;
return (-0.5*(nobs*LN2PI+ double(lik))*SCALE);
}
L    C++ Code provided by Gilles Teyssière (arch-min.cc)

#include"arch.h"
#include"GLRstat.h"

extern "C"
{
    double sqrt(double);
    double fabs(double);
}

#define NUM_REC_VER 1

static const int ITMAX(200); // MAXIMUM NUMBER OF ITERATIONS
static const double EPS(10*EPSMCH);
static const double TOLX(4*EPSMCH);
static const double MEPSMCH(100*EPSMCH);
static const double STPMX(10.0);

template <class T>
    static inline T FMAX(T a,T b) {return a>b ? a:b;};

template <class T>
    static inline T SQR(T a) {return a*a;};

template <class T>
    T ** matrix (T **,int nr1,int nrh,int ncl,int nch)
    
    {
        T ** m;
        if (!((m = new T *[nrh+1])))
            nrerror("ALLOCATION FAILURE 1 IN MATRIX");
        for (int i=1; i<=nrh;i++)
            if (!((m[i] = new T [nch+1])))
                nrerror("ALLOCATION FAILURE 2 IN MATRIX");
        return m;
    }
template <class T>
  void free_matrix(T **m, int nrl, int nrh, int ncl)
  {
    for (int i=nrh; i>=1; i--)
      delete [] m[i];
    delete [] m;
  }

extern void nrerror(char* text);
extern void disp (int, double *, int);

void near_Jnsrch(double* xold, double fold, double* g,
                double* p, double* x, double& f, double stpmax, ARCH &,
                double (*func)(double*, ARCH&), int &cv);

void dfpmin(double *p, ARCH& msc, double &fret, double(*func)(double *,ARCH &),
             void(*dfunc)(double*,double*,double(*f)(double*,ARCH &), ARCH &),
             double stpmax, double(*dfunc)(double*,double*,double(*f)(double*,ARCH &),
             ARCH &), ARCH &, double));

/// ALGORITHME DE MINIMIZATION

void dfpmin(double *p, ARCH& msc, double(*func)(double *,ARCH &),
             void(*dfunc)(double*,double*,double(*f)(double*,ARCH &), ARCH &),
             ARCH &, double))
{
  int j, i, its, cv, n(msc.nvar());
  double fae, fad, fac, temp, test, sumdg, sumxi, den, fp, sum(0.0);
  double stpmax(0.1);
  double *dg, *g, *hdg, *pnew, *xi;
  long double **hessin;
  double gtol(msc.gtol);
  long double sum1, sum2, sum3, sum4;

  dg = new double [n+1];
g = new double [n+1];
hdg = new double [n+1];
pnew = new double [n+1];
xi = new double [n+1];
hessin = matrix(hessin,1,n,1,n);

fp = (*func)(p,msc);
(dfunc)(p,g,func,msc,fp);
for (i=1;i<=n;++i)
{
    for(j=i+1;j<=n;++j)
        *(*hessin+j)+i= *(*hessin+i)+j = 0.0;
    *(*hessin+i)+i = 1.0;
    xi[i] = - g[i];
}
double fret(0);

for (its=1;its<=ITMAX;its++)
{
    #if DISPLAY
        disp (its,p,n);
        //disp (its,g,n);
    #endif
    cv = 1;
    msc.cv(0);
    near_insrch(p,fp,g,xi,pnew,fret,stpmax,msc,func,cv);
    if (cv == -1)
    {
        // DESALLOCATION TOTALE ET RETOUR
        msc.cv(1);
        free_matrix(hessin,1,n,1);
        delete [] xi;
        delete [] g;
        delete [] dg;
        delete [] hdg;
        delete [] pnew;
        return;
    }
    for (i=1;i<=n;i++)
    {
        xi[i] = pnew[i] - p[i];
    }
p[i] = pnew[i];
}

fp = fret;
test = 0.0;
#endif

for (i=l;i<=n;i++)
{
    // temp = fabs(xi[i])/FMAX(fabs(p[i]),1.0);
    temp = fabs(xi[i])/FMAX(fabs(p[i]),0.1);
    //temp = fabs(xi[i])/fabs(p[i]);
    if (temp > test) test = temp;
}

if (test < TOLX)
{
    //DESALLOCATION TOTALE ET RETOUR
    free_matrix(hessin,1,n,1);
    delete [] xi;
    delete [] g;
    delete [] dg;
    delete [] hdg;
    delete [] pnew;
    return;
}
#endif

for (i=1;i<=n;i++)
dg[i]= g[i];
(*dfunc)(p,g,func,msc,fp);
test= 0.0;
// TEST DE CONVERGENCE DU GRADIENT VERS ZERO
#if NUM_REC_VER == 1
    den= FMAX(fret,double(1.0));
    for (i=1;i<=n;++i)
    {
        // temp=fabs(g[i]) * FMAX(fabs(p[i]),1.0)/den;
        temp=fabs(g[i]) * FMAX(fabs(p[i]),0.1)/den;
        //temp=fabs(g[i]) * fabs(p[i])/den;
        if (temp > test) test=temp;
    }
#endif

71
#else
den = msc.read_nb_obs() - TRONC_PS;
for (i=1;i<=n;i++)
{
    temp = fabs(g[i])/den;
    if (temp > test) test = temp;
}
#endif
if (test < gtol)
{
    // DESALLOCATION TOTALE ET RETOUR
    free_matrix(hessin,1,n,1);
    delete [] xi;
    delete [] g;
    delete [] dg;
    delete [] hdg;
    delete [] pnew;
    return;
}
for (i=1;i <= n;i++) dg[i] = g[i] - dg[i];
for (i=1;i <= n;i++)
{
    //hdg[i]=0.0;
    sum1 = 0.0;
    for (j=1;j<=n;j++)
        sum1 += (*(*((hessin+i)+j))* dg[j];
    hdg[i] = double(sum1);
}
//fac=fae=sumdg=sumxi=0.0;
sum1 = sum2 = sum3 = sum4 = 0;
for (i=1;i<=n;i++)
{
    sum1 += dg[i] * xi[i];
    sum2 += dg[i] * hdg[i];
    sum3 += SQR(dg[i]);
    sum4 += SQR(xi[i]);
}
fac = double(sum1);
fae = double(sum2);
sumdg = double(sum3);
sumxi = double(sum4);
if (fac*fac > EPS*sumdg*sumxi)
{
    fac = 1.0/fac;
    facd = 1.0/fae;
    for (i=1;i<=n;i++)
        dg[i] = fac* xi[i] - fad* hdg[i];
    for(i=1;i<=n;i++)
    for(j=i;j<=n;j++)
        (*(*(hessin+j)+i) += (*(*(hessin+i)+j) + fae * dg[i] * dg[j];
    }
}
for(i=1;i <= n;i++)
{
    sum1 = 0.0;
    for (j=1;j <= n;j++)
        sum1 -= *(hessin+i)+j)* g[j];
    xi[i] = double(sum1);
}
}

// DESALLOCATION GLOBALE ET RETOUR
free_matrix(hessin,1,n,1);
delete [] xi;
delete [] g;
delete [] dg;
delete [] hdg;
delete [] pnew;
}

/////////////////////////////////////////////////////////////////////////
// FONCTION LINESEARCH
/////////////////////////////////////////////////////////////////////////
static const double ALF(1.0e-4);
static const double TOL_X(1.0e-7);
void near_Jnsrch(double* xold, double fold, double* g, double*p, double* x, 
                double& f, double stpmax, ARCH & msc,
                double (*func)(double*,ARCH &), int &cv)
{
    int i, n(msc.nvar());
    long double a,alam2, alamin,b,disc,f2,fold2,rhsl,rhs2,slope,sum;
    long double temp, test,tmplam;
    for (sum=0.0,i=1;i<=n;i++)
        sum += p[i] * p[i];
    sum = sqrt(sum);
    if (sum>stpmax)
        for (i=1;i<=n;i++) p[i] *= stpmax/sum;
    for (slope=0.0,i=1;i<=n;i++)
        slope += g[i] * p[i];
    test = 0.0;
    for (i=1;i<=n;i++)
    {
        temp = fabs(p[i])/FMAX(fabs(*(xold+i)),1.0);
        if (temp > test) test =temp;
    }
    alamin = TOLJC/test;
    long double alam = 1.0;
    for (;;)
    {
        for (i=1;i<=n;i++) x[i] = xold[i] + alam * p[i];
        f = (*func)(x,msc);
        if (alam<alamin)
        {
            for (i=1;i<=n;i++) x[i] = xold[i];
            return;
        }
        else if (f<=fold+ALF*alam*slope) return;
        else
        {
            if (alam == 1.0)
                tmplam = -slope/(2.0*(f-fold-slope));

34
else
{
    rhs1=f-fold-alam*slope;
    rhs2=f2-fold2-alam2*slope;
    a=(rhs1/(alam*alam)-rhs2/(alam2*alam2))/(alam-alam2);
    b=(-alam2*rhs1/(alam*alam)+alam*rhs2/(alam2*alam2))/(alam-alam2);
    if (a==0.0) tmplam = -slope/(2.0*b);
    else
    {
        disc = b*b-3.0*a*slope;
        if (disc<0.0)
        {
            if (disc < -1.0e-4)
            {
                cv = -1;
                return;
            }
            else disc =0;
        }
        tmplam= (-b+sqrt(disc))/(3.0*a);
    }
    if (tmplam > 0.5*alam) tmplam = 0.5*alam;
}
alam2 = alam;
f2 = f;
fold2 = fold;
alam = FMAX(double(tmplam),double(0.01*alam));
}

END OF MODULE ARCH-min.cpp
extern "C" {
  long double pow(double,double);
  double sqrt(double);
  double fabs(double);
};

#include"arch.h"
#include"GLRstat.h"
#include"math.h"
template <class T>
inline T FMAX(T a,T b) {return a>b ? a:b;};

const double PO_EPS(1.0/3.0);
const double EPSILON1(sqrt(EPSMCH));
const double EPSILON2(pow(EPSMCH,PO_EPS));

extern void nrerror(char*);

template <class T>
  T ** matrix (T **,int nrl,int nrh,int ncl,int nch)
  {
    T **m ;
if (!(m = new T *[nrh+l]))
    nrerror("ALLOCATION FAILURE 1 IN MATRIX");
for (int i=1; i<= nrh;i++)
    if (!(m[i] = new T [nch+l]))
        nrerror("ALLOCATION FAILURE 2 IN MATRIX");
return m;

/*====================================================================*/
// TEMPLATE FUNCTION FREE_MATRIX, WHICH RELEASES THE MEMORY ALLOCATED BY MATRIX
/*====================================================================*/

#include <iostream>

#include "utilities.h"

void free_matrix(T **m, int nrl, int nrh, int ncl)
{
    for (int i=nrh; i>= 1; i--)
        delete [] m[i];
    delete [] m;
}

void num_deriv(double *x, double *gr, double (*f)(double*, ARCH &), ARCH & msc);

/*====================================================================*/
// NEAR FUNCTIONS
/*====================================================================*/

void covar_h(double *x, double **h, ARCH &msc,
              double (*f)(double*, ARCH &), int &pd);
void num_hessian(double *x, double **a, ARCH &msc,
                 double (*f)(double*,ARCH &) );
void choldc(double **a, int n, double *p, int &ty);
void cholinv(double **a, int n, int &pd);

#if G_METHOD == 0
/*====================================================================*/
// CENTRAL NUMERICAL DERIVATIVES
/*====================================================================*/

void num_deriv(double *x, double *gr, double(*f)(double*,ARCH &),

ARCH & msc, double fx)
{
int i, j, n(msc.nvar());
double h, temp, *tf_x, *tb_x;

tf_x = new double [n+1];
tb_x = new double [n+1];
for (i=1; i<=n; i++)
{
    double s_limit(0.01);
    double scale = FMAX(fabs(x[i]), s_limit);
    temp = x[i] + (h=EPSILON2*scale);
    h = temp - x[i];
    for (j=1; j<=n; j++)
    { 
        tb_x[j] = tf_x[j] = x[j];
        tf_x[i] = x[i] + h;
        tb_x[i] = x[i] - h;
        gr[i] = ((*f)(tf_x,msc)- (*f)(tb_x,msc))/(h+h);
    }
}
delete [] tf_x;
delete [] tb_x;
}

#else

void num_derv (double *x, double *gr, double(*f)(double*,ARCH &
), ARCH & msc, double fx)
{
int i, j, n(msc.nvar());

double h, temp, *tf_x;
.tf_x = new double [n+1];

#endif
for (i=1; i<=n; i++)
{
    double scale = FMAX(fabs(x[i]),0.01);
    temp = x[i] + (h=EPSILON1*scale);
    h = temp - x[i];
    for (j=1; j<=n; j++)
        tf.x[j] = x[j];
    tf.x[i] = x[i] + h;
    gr[i] = ((*f)(tf.x,msc)-fx)/h;
}
delete [] tf.x;
}
#endif
#if HCCM == 1

void ARCH::hessian(double *x, double(*func)(double*,ARCH &))
{
    double **inv_hes;
    int n(NVAR), pd(1);

    inv_hes = matrix(inv_hes,1,n,1,n);
    covar_h (x,inv_hes,*this,func,pd);
    if (pd == 0) pos_def = 0;
    else pos_def = 1;
    free_matrix(inv_hes,1,n,1);
}

#endif
#if HCCM == 1

// FUNCTION covar, CALCULATES THE HC COVARIANCE MATRIX OF THE ML ESTIMATES

void ARCH::hessian(double *x, double(*func)(double*,ARCH &))
{
    double **inv_hes;
    int n(NVAR), pd(1);

    inv_hes = matrix(inv_hes,1,n,1,n);
    covar_h (x,inv_hes,*this,func,pd);
    if (pd == 0) pos_def = 0;
    else pos_def = 1;
    free_matrix(inv_hes,1,n,1);
}

// FUNCTION covar_h, CALCULATES THE INVERSE OF MINUS THE HESSIAN MATRIX

79
void covar_h(double *x, double **h, ARCH& msc,  
  double (*f)(double*, ARCH&), int &pd)  
{
  int n(msc.nvar());

  num_hessian (x,h,msc,f);
  chol_inv(h,n,pd);
}

///////////////////////////////////////////////////////////////////////////////////  
// FUNCTION num_hessian, COMPUTES MINUS THE NUMERICAL HESSIAN MATRIX  
// BY CENTRAL DIFFERENCE METHOD  
///////////////////////////////////////////////////////////////////////////////////
void num_hessian(double *x, double **a, ARCH &msc,  
  double (*f)(double*,ARCH &) )  
{
  int i,j,k, n(msc.nvar()), T(msc.read_nb_obs());
  long double sum, h;

  tf = new double [n+1];
  tb = new double [n+1];
  tff = new double [n+1];
  tbf = new double [n+1];
  tbb = new double [n+1];
  md = f(x,msc);
  for (i=1;i <= n; i++)  
  {  
    double s_limit = 0.01;
    double scale = FMAX(fabs(x[i]),s_limit);
    temp = x[i] + (h1 = EPSILON2*scale);
    h1 = temp - x[i];
    for (k=1;k <= n;k++)  
    {
      
      
    }  
  }  
}
if (k==i)
{
  tf[k] = x[k] + h1;
  tb[k] = x[k] - h1;
}
else tf[k] = tb[k] = x[k];
}
for (j=i;j <= n;j++)
{
  scale = FMAX(fabs(x[j]),s_limit);
  temp = x[j] + (h2 = EPSILON2*scale);
  h2 = temp - x[j];
  for (k=1;k <= n;k++)
  {
    if (k==j)
    {
      tff[k] = tf[k] + h2;
      tfb[k] = tf[k] - h2;
      tbf[k] = tb[k] + h2;
      tbb[k] = tb[k] - h2;
    }
    else
    {
      tff[k] = tfb[k] = tf[k];
      tbf[k] = tbb[k] = tb[k];
    }
  }
  ff = (*f)(tff,msc);
  if (i==j) bf = fb = md;
  else
  {
    fb = (*f)(tfb,msc);
    bf = (*f)(tbf,msc);
  }
  bb = (*f)(tbb,msc);
  h = 4*h1*h2*T;
  *(a+i)+j) = *(a+j)+i) = double(sum = (ff-bf-fb+bb)/ h);
}

81
FUNCTION choldc
CHOLESKY DECOMPOSITION OF A MATRIX
void choldc(double **a, int n, double *p, int & ty) {
    int i, j, k;
    double sum;

    for (i=1;i <= n;i++)
    {
        for (j=i;j <= n;j++)
        {
            for (sum= a[i][j], k=i-1;k >= 1;k--)
                sum -= a[i][k] * a[j][k];
            if (i == j)
            {
                if (sum <= 0.0)
                {
                    //Matrix not PD
                    ty = 1;
                    return;
                }
                p[i] = sqrt(sum);
            }
            else
                a[j][i] = (sum/p[i]);
        }
    }
}

82
FUNCTION cholinv
INVERSION OF A POSITIVE DEFINITE MATRIX BY CHOLESKY DECOMPOSITION

void cholinv(double **a, int n, int &pd)
{
    double *p, **at;
    double sum;
    int i, j, k, ty(0);

    at = matrix(at,1,n,1,n);
    p = new double [n+1];

    choldc(a,n,p,ty);
    if (!ty)
    {
        for (i=1;i <= n;i++)
            for (j=i+1;j <= n;j++) a[i][j] = 0;
        for (i=1;i <= n;i++)
            { 
                a[i][i] = 1.0 / p[i];
                for (j=i+1;j <= n;j++)
                    { 
                        sum = 0.0;
                        for (k=i;k < j;k++)
                            sum -= a[j][k] * a[k][i];
                        a[j][i] = (double(sum) / p[j]);
                    }
            }
        for (i=1;i <= n;i++)
            for (j=i;j <= n;j++)
                { 
                    at[j][i] = 0.0;
                    long double sum1(0.0); 
                    for (k=1;k<=n;k++)
                        ...}
```c
{
    sum1 += a[k][i] * a[k][j];
    at[i][j] = double(sum1);
}
}

for (i=1;i<=n;i++)
    for (j=i;j<=n;j++)
        a[i][j] = a[j][i] = at[i][j];
else
{
    for (i=1;i<=n;i++)
        for (j=i;j<=n;j++)
            a[i][j] = a[j][i] = 1.0;
    pd = 0;
}
free_matrix(at,1,n,1);

hhspace*.3in hhspace*.3in delete [] p;
}

#endif

// END OF MODULE ARCH-hdnum.cpp ////////////////
N C++ Code provided by Gilles Teyssière (normal.cc)

///////////////////////////////////////////////////////////////////////////////////////////
// This routine returns normal deviates
///////////////////////////////////////////////////////////////////////////////////////////

extern "C"{
    double sqrt(double);
    double log(double);
    double floor(double);
}

double ran2(long &idum);
double ran3(long &idum);
double gasdev(long &idum, long &idum1);

const double R23 = 2.0/3.0;
const double R43 = 2.0*R23;

//long idum = -1;
//long idum1 = -2;

// IN THE FIRST CALL, TWO NEGATIVE NUMBER idum and idum1 ARE USED AS ARGUMENTS

// FOR INITIALISING THE PROCESS. IN THE SUBSEQUENT CALLS OF THIS ROUTINE, idum1 and idum

// WILL KEEP THEIR UPDATED VALUES .

//THIS ROUTINE RETURNS A GAUSIAN DEVIATE N(0,1)

double gasdev(long &idum, long &idum1)
    {
        static int iset(0);
        static double gset;
        return gset;
    }
double fac, rsq, v1, v2;
double alpha = ran3(idum)+ran2(idum1);

if (iset == 0)
{
    do
    {
        if (alpha < R23)
        {
            v1 = 2.0*ran3(idum)-1.0;
            v2 = 2.0*ran3(idum)-1.0;
        }
        else if (alpha < R43)
        {
            v1 = 2.0*ran3(idum)-1.0;
            v2 = 2.0*ran2(idum1)-1.0;
        }
        else
        {
            v1 = 2.0*ran2(idum1)-1.0;
            v2 = 2.0*ran2(idum1)-1.0;
        }
        rsq = v1*v1 + v2*v2;
    }
    while (rsq >= 1.0 || rsq == 0.0);
    fac = sqrt(-2.0*log(rsq)/rsq);
    gset = v1*fac;
    iset =1;
    return v2*fac;
}
else
{
    iset = 0;
    return gset;
}
This routine returns a rv uniformly distributed
Ref: Numerical Recipes (1992), p 280
KNUTH'S ALGORITHM

#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)

double ran3(long &idum)
{
    static int inext,inextp;
    static long ma[56];
    static int iff(0);
    long mj, mk;
    int i, ii, k;

    if (idum < 0 || iff == 0)
    {
        iff=1;
        mj=MSEED-(idum < 0 ? -idum : idum);
        mj %= MBIG;
        ma[55]=mj;
        mk=1;
        for (i=1;i<=54;i++)
        {
            ii=(21*i) % 55;
            ma[ii]=mk;
            mk=mj-mk;
            if (mk < MZ) mk += MBIG;
            mj=ma[ii];
        }
        for (k=1;k<=4;k++)
        for (i=1;i<=55;i++)
        {
            ma[i] -= ma[1+(i+30) % 55];
            if (ma[i] < MZ) ma[i] += MBIG;
        }
    }
    return mj * FAC;
}

inext=0;
inextp=31;
idum=1;
}
if (++inext == 56) inext=1;
if (++inextp == 56) inextp=1;
mj=ma[inext]-ma[inextp];
if (mj < MZ) mj += MBIG;
ma[inext]=mj;
return mj*FAC;
}
#undef MBIG
#undef MSEED
#undef MZ
#undef FAC
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 2.22046e-15
#define RNMX (1.0-EPS)

double ran2(long &idum)
{
    int j;
    long k;
    static long idum2(123456789);
    static long iy(0);
static long iv[NTAB];
double temp;

if (idum <= 0)
{
    if (-(idum) < 1) idum=1;
    else idum = -(idum);
    idum2=(idum);
    for (j=NTAB+7;j>=0;j--)
    {
        k=(idum)/IQ1;
        idum=IA1*(idum-k*IQ1)-k*IR1;
        if (idum < 0) idum += IM1;
        if (j < NTAB) iv[j] = idum;
    }
    iy=iv[0];
}

k=(idum)/IQ1;
idum=IA1*(idum-k*IQ1)-k*IR1;
if (idum < 0) idum += IM1;
k=idum2/IQ2;
idum2=IA2*(idum2-k*IQ2)-k*IR2;
if (idum2 < 0) idum2 += IM2;
j=iy/NDIV;
iy=iv[j]-idum2;
iv[j] = idum;
if (iy < 1) iy += IMM1;
if ((temp=AM*iy) > RNMX) return RNMX;
else return temp;

#undef IM1
#undef IM2
#undef AM
#undef IMM1
#undef IA1
#undef IA2
#undef IQ1
#undef IQ2
#undef IR1
#undef IR2
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX
References


