Consistent estimation of breakpoints in time series, with application to wavelet analysis of Citigroup returns

Leigh Roberts
The Working Paper series is published by the School of Economics and Finance to provide staff and research students the opportunity to expose their research to a wider audience. The opinions and views expressed in these papers are not necessarily reflective of views held by the school. Comments and feedback from readers would be welcomed by the author(s).

Further enquiries to:
The Administrator
School of Economics and Finance
Victoria University of Wellington
P O Box 600
Wellington 6140
New Zealand

Phone: +64 4 463 5353
Email: alice.fong@vuw.ac.nz
Consistent estimation of breakpoints in time series, with application to wavelet analysis of Citigroup returns

Leigh Roberts∗

February 6, 2014

Abstract

Simple and intuitive non-parametric methods are provided for estimating variance change points for time series data. Only slight alterations to existing open-source computer code applying CUSUM methods for estimating breakpoints are required to apply our proposed techniques. Our approach, apparently new in this context, is first to define two artificial time series of double the length of the original by reflective continuations of the original. We then search for breakpoints forwards and backwards through each of these symmetric extensions to the original time series.

A novel feature of this paper is that we are able to identify common breakpoints for multiple time series, even when they collect data at different frequencies. In particular, our methods facilitate the reconciliation of breakpoint outputs from the two standard wavelet filters.

Simulation results in this paper indicate that our methods produce accurate results for time series exhibiting both long and short term correlation; and we illustrate by an application to Citigroup stock returns for the last thirty years.

Key words:
Breakpoint, change point, variance change point; model-free, non-parametric; R programming suite, R package waveslim; wavelets, DWT (discrete wavelet transform), MODWT (maximal overlap discrete wavelet transform), MRA (multiresolution analysis); CUSUM (cumulative sum of squares); cluster analysis.

∗School of Economics and Finance, Victoria University, Wellington
Email: leigh.roberts@vuw.ac.nz
Contents

1 Introduction 4

2 Preliminary 9

  2.1 Literature review ................................................. 9
  2.2 Mean correction .................................................. 12
  2.3 Wavelets .......................................................... 14

3 The CUSUM Method 18

4 Searching for breakpoints 21

  4.1 Methodology of searching for breakpoints in mult.loc ....... 22
  4.2 The ancillary role of MODWT in mult.loc ................. 23

5 An extended example with the normal distribution 25

  5.1 CUSUM for the original time series \( z \neq 0 \), and its reversal and reflections ............................................. 26
  5.2 The normal distribution: breakpoints of 4th wavelet details by CUSUM applied to wavelet coefficients ..................... 36
      5.2.1 Translation of wavelet coefficient breakpoints into the original time scale ........................................... 36
      5.2.2 Application of testing.hov to produce breakpoints for the 4th detail of the MRA of \( z \neq 0 \) ....................... 37
  5.3 The normal distribution: direct CUSUM for the 4th MRA detail 44
6 Examples with heavy-tailed and long memory distributions

6.1 Simulation of the $t$ distribution with 4 degrees of freedom

6.2 Simulation of a long memory process: the Hosking approach

7 Citigroup returns over the last 30 years

7.1 Comparison of three stocks

7.2 Citigroup

7.2.1 History

7.2.2 Citigroup daily returns and its MRA

7.2.3 Citigroup breakpoints

7.2.4 CUSUM applied directly to the wavelet details

8 Conclusion

Appendix A

Appendix B
1 Introduction

The incentive to write this paper came from a desire to estimate variance change points for stock returns, which desire sprang in turn from the central importance of volatility in finance. We use CUSUM (cumulative sums of squares) methodology to identify points of a time series at which the series jumps or moves unexpectedly, and do not distinguish between labels such as breakpoints, change points, variance change points, volatility change points, etc.

In the financial context, volatility is defined as the standard deviation of returns on financial assets (e.g., Taylor (2005, Ch. 8)), occasionally defined alternatively as the variance rather than the standard deviation. Theorists in finance want to keep their cake and to eat it as well: volatility is to be varying over time, and most models permit gradually changing volatility, frequently within one of the GARCH family of models; but also desired is the feature of clustering volatility, so that periods of high volatility will tend to alternate with periods of low volatility. In addition, volatility is well known to have long memory properties, which needs to be allowed for in modelling. These perhaps conflicting goals notwithstanding, it is clearly of prime importance to identify breakpoints in financial returns, and particularly variance or volatility change points. The extent to which such breakpoints delineate volatility clusters is a subject for further research.

Volatility in finance is intrinsically model dependent, as noted in Fryzlewicz (2013a), i.a. In univariate time series practice, however, realised values of the volatility are often simply taken to be the absolute values of the return (e.g., Gencay, Selcuk & Whitcher (2002), Taylor (2005)). While this is tantamount to estimating the standard deviation from a sample of one, with an expected value of $0.8 \sigma$ for i.i.d. zero mean normal $\mathcal{N}(0, \sigma^2)$ data, one substantial advantage of adopting this approach is that realised values of the random volatility are no longer model dependent. It is in this spirit that we restrict ourselves to non-parametric techniques for estimating breakpoints of time series, and such techniques can be expected to find applications outside the financial sphere.
More generally, finance practitioners wish to investigate the behaviour of stock returns at varying investment horizons, for which wavelets are well suited. Wavelets break a signal down into components restricted to particular frequency bands. Speculators and those hedging over short time horizons will be interested in the behaviour of returns over short horizons, or low levels of wavelet analysis, viz. those at the highest frequency. Those wishing to hedge over say 3 month time periods may not be overly concerned at short run trends in prices and returns; and long run investors such as life insurance companies and pension funds will not usually be interested in short run behaviour in the markets unless there are ramifications for the longer term. In other words, one does not just want to identify breakpoints in a time series of stock returns; one is also interested in simultaneously extracting breakpoints from the components of that return series at the different wavelet levels. Hence one further reason at least for a non-parametric approach to identifying breakpoints: it is hard enough to find and fit a sensible model for behaviour of returns over a given time horizon; but trying to model simultaneously the data and its MRA extending over several frequency bands would be a formidable task.

The basic technique to be applied in this paper is the CUSUM (cumulative sum of squares) method for finding breakpoints, applied in a slightly unorthodox manner to data which is input directly, then in the reverse order; after which the data is reflected symmetrically, and again input forwards and backwards.

Inputting the data in reverse order is natural enough. For general searches for breakpoints, one has to be some distance past the point of change in order to realise that the variance has changed; running data backwards and forwards through the programme, then, may yield a better indication of where change points lie, since the algorithms involved do not operate symmetrically. At first blush, however, the further step of inputting symmetric extensions of the data forwards and backwards into the computer sounds nonsensical, because of course the same output would be produced. All that is meant in fact is that the output from this procedure is interpreted in two distinct ways.
The intent is that each of these types of computer runs will produce fresh estimates of breakpoints, some of which will be easily meshed with breakpoints from other of the four basic methods, but some of which may appear to be ‘new’ breakpoints. The final choice of breakpoints should in any case be validated by looking at the putative results on a plot of the original time series, and at that stage one might well decide to strengthen the criteria for choosing breakpoints in order to produce fewer of them, or merge two into one; or relax the criteria in order to produce more breakpoints. A natural technique to use in identifying breakpoints is cluster analysis (e.g., Kendall (1965), Johnson (1967)); and while we do apply this tool, it must be admitted that the whole process of identifying breakpoints is more of an art than a science, especially in a non-parametric context.

The methods adumbrated in this paper for choosing breakpoints possess the additional advantage that little additional work is required to adapt them to estimating joint breakpoints for several time series, some of which could gather data at different frequencies. Our simple methods set out below would for instance be suitable for considering joint timings of breakpoints for daily returns and weekly returns over the same period. It is then a short further step to estimating common breakpoints for different frequency bands, or wavelet levels, simultaneously.

Finally, a further important advantage is that relatively little coding needs to be done to implement the methods in this paper. While the CUSUM technique is well-known, and presumably has much source code already available from many sources, our approach to simultaneous estimation of breakpoints for returns and their wavelet components is readily effected in the computer programme suite R.

The programming environment R is open source, and easily available free of charge from the internet (CRAN (2013)). The two packages that we use in R are waveslim (Whitcher (2013)) for the CUSUM calculations; and FactoClass (Pardo, del Campo & Torres (2013)) for cluster analysis employed as a possible aid in sorting out which putative breakpoints should be merged and which separated as ‘gen-
uine' individual breakpoints.

Within R the waveslim package can be downloaded by the commands:

```r
install.packages("waveslim")
library(waveslim)#to access waveslim in R
```

and similarly for the package FactoClass.

In order to use R in ‘batch mode’, one can save code in a file and ‘source’ the file in R. Alternatively, it is often easier to ‘cut and paste’ code from ASCII, text or pdf files on a screen into an R operating window.

The waveslim package in R largely contains the code that Whitcher used in the book Gencay et al. (2002), and in particular contains two functions which suit our purpose admirably, all things considered: while they are designed for finding consistent breakpoints for the two principal wavelet transforms, it is easy to adapt them to our more general purposes. Finding breakpoints for wavelet transforms makes use of the function testing.hov (for ‘testing for homogeneity of variance’), which possesses two aims in life: its first purpose is to remove some preliminary wavelet coefficients which are dependent on boundary conditions, these being considered unreliable for the purpose of estimating breakpoints; and whose secondary goal in life is to hand the underlying CUSUM calculations over to the function mult.loc, also in the waveslim package, which does all the work. For those wishing to estimate breakpoints in time series and having no interest in applying wavelets, simple adaptations of the input into the latter function mult.loc suffice to estimate joint breakpoints for multiple time series, and the former function testing.hov can be disregarded.

Use of cluster analysis in sorting out which breakpoints to accept and which to merge or reject may or may not be instructive, and should in any case be complemented by visual verification on a plot of the data. A simple means of applying cluster analysis is to use the function kmeans, available in standard R downloaded from the internet; more useful in our context however would seem to be the function
kmeansW, available in the R package FactoClass, which forms clusters allowing for weights to be assigned to the individual elements. Required input to both of the functions kmeans and kmeansW is the number of clusters to be formed, highlighting the subjective nature of the whole exercise.

The plan for the paper is as follows.

The material in §2 is preliminary, with the first part containing a literature review; there follows a discussion on the centring of variances in this paper, or rather their non-centring, since we do not correct data for the mean when estimating variances. A short discussion gives some background on wavelets, and examples of the way in which wavelets decompose a signal into ‘details’ and ‘smooths’ at various frequency bands, illustrated by showing the Multiresolution Analyses (MRAs) for returns on Exxon-Mobil stock and Citigroup stock in Figures 1 and 2 on pp. 15 and 17 respectively.

Proceeding to numerical examples, §3 sets out the basic mathematics of the CUSUM methodology, illustrating it with a simple time series of length 10, in Figure 3 on p. 18. Another example of length 10, but this time containing reflected data, reveals a potential snag with symmetrically extending our data before applying the CUSUM methods for finding breakpoints. The problem is readily overcome, and this example is illustrated in Figure 4 on p. 22.

We proceed to estimate breakpoints for more realistic examples, adopting in turn the following approaches.

1. CUSUM applied to the original time series;
2. CUSUM applied to DWT and MODWT coefficients from a wavelet transform of the original time series;
3. CUSUM applied to wavelet details in the Multiresolution Analysis of the original time series.

The first of these approaches directly utilises the programme mult.loc for the CUSUM analysis of the original data. The second utilises testing.hov, firstly to calculate wavelet transforms of the data, which
in turn are passed to \texttt{mult.loc} for the CUSUM analysis. Mirroring the first approach, the third approach again applies \texttt{mult.loc} directly to wavelet details, the fourth MRA detail being chosen as illustrative of our techniques.

Each of the above 3 approaches contains 4 substeps, involving forward and reverse passes of the data itself and its reflections, as set out in some detail in §5 on p. 25 in the context of our first example, viz. simulated independent zero mean normal variates, with variances constant within blocks of the data.

We proceed in §6 to identify breakpoints, firstly for simulated thicker tailed distributions, viz. a \(t\) distribution with low degrees of freedom, but still using independent realisations with piece-wise constant variance; and secondly for a simulated long memory process. Our methodologies stand up well to the more stringent tests posed by the latter two simulations, showing the utility of estimating breakpoints by several methods simultaneously.

In §7 we estimate breakpoints for daily returns for Citigroup over the last 30 years, along with breakpoints for the wavelet details in the multiresolution analysis (MRA); and §8 offers a short conclusion.

\section{Preliminary}

\subsection{Literature review}

Structural changes in time series can assume the form of level shifts, variance shifts, outliers or presumably many other types of irregularities which may impede effective modelling. Whatever the form of structural breakpoints, accurate identification of their location is clearly an important endeavour, as is the clarification of the impact of such changes on the long term behaviour of the time series. This is especially true for any sort of regime switching model.
The various types of breakpoints are moreover not mutually exclusive. Ureche-Rangau & Speeg (2011) are not the first authors to point out that level shifts and variance change points are likely to occur together. In the same vein, Tsay (1988) indicates a more exhaustive, and exhausting, taxonomy, regarding outliers as a ‘temporary’ species of change point, in contrast to the more permanent level shifts and variance change points; he further differentiates between different types of outliers, and permanent and transient level shifts. In Tsay (1987), breakpoint definitions are put into operational terms within a stationary time series framework; in the later paper Tsay (1988) translates them into alternative types of ‘variance ratios’.

Application of CUSUM variables to indicate variance change points appears to have originated in Brown, Durbin & Evans (1975), who applied known results on distributions of periodogram estimators to find variance change points. This was followed up in Hsu (1977), who used Edgeworth expansions and Dirichlet distributions to approximate the distribution of CUSUM variables. Wichern, Miller & Hsu (1976) separated data into blocks and searched for variance breaks in between the blocks. An in-depth discussion of these and earlier developments in estimating breakpoints can be found in Whitcher, Byers, Guttorp & Percival (1998), while a more recent summary of the literature of breakpoint modelling appears in Ureche-Rangau & Speeg (2011).

Current research into breakpoint determination is partly focused on high dimensional data. Cho & Fryzlewicz (2013), for instance, aim to modify CUSUM in high dimensional searching by ‘Sparsified Binary Segmentation’, noting that CUSUM works much less well in higher dimensions, and admitting only CUSUM statistics which exceed a certain threshold. Our approach is hardly high dimensional in the sense of these authors; and we simply apply univariate CUSUM methods to our related time series, linking breakpoints by cluster analysis.

Whereas statisticians tended to non-parametric or pure time series techniques, econometricians estimated breakpoints in the context of
assumed models, generally assorted types of linear models or dynamic economic models. Bai & Perron (2003), for instance, appeal to asymptotic theory for general variants on the linear model, but assume that level shifts and variance change points occur together; while Bai, Lumsdaine & Stock (1998) utilise more elaborate dynamic economic models, aiming to identify common breakpoints.

Econometricians’ models are relatively tightly specified, especially as regards the assumed behaviour of the mean value, because economists do not often have so much data: quarterly or monthly data even over long periods does not necessarily translate into much data for the purposes of establishing structural breakpoints of a rather ‘portmanteau’ nature. Additionally, economists often have a reasonable idea of where structural breaks happened, and the sort of breaks that did in fact occur. They are perhaps then entitled to work with relatively finely wrought models.

The situation may be somewhat more favourable for those working with financial data. On the one hand, accurate estimates of volatility (the standard deviation of returns) is vital in order to gauge the risk of investments and price market derivatives (see for instance Jawadi & Ureche-Rangau (2013)); on the other hand, however, financial engineers often have access to extremely voluminous data, at least as long as it is simply concatenated: daily returns over a few years amounts to a lot of data from which to estimate breakpoints; and five minute returns on exchange traded stocks even over a few weeks can give rise to a large sample. To be sure, financial engineers also often have some idea of when and why structural breaks have occurred in their data; but identifying those breakpoints to the finest data interval available is challenging, especially given the distortionary impact of data concatenation.

In the testing.hov function in the Waveslim package in R, Whitcher et al. (1998) utilise the CUSUM search method set out in Inclan & Tiao (1994) to the two principal wavelet filters, the DWT (discrete wavelet transform) and the MODWT (maximal overlap discrete wavelet transform). The mechanics of the search are duplicated for the DWT and the MODWT: the former filter allows for correct
inference concerning the presence of breakpoints, but the latter is
more readily interpretable. More precisely, the former transform
results in a truncated uncorrelated series, for which the connection
with the original time axis is somewhat tenuous; the latter preserves
a closer relationship with the original time variable, but is autocorre-
related.

An application of this tandem approach is reported in Gencay et al.
(2002, §7.3), who estimate breakpoints for volatility of IBM stock
returns; it is the code for this application that is written up in
testing.hov and mult.loc (Whitcher (2013)), and which we utilise in
this paper, with a broadly similar application in mind. Both of these
functions are listed in Appendix B on p. 80, since we refer to their
structure at odd times in this report.

2.2 Mean correction

Despite our initial claim to be seeking ‘variance change points’, we
work with sums of squares in a CUSUM (cumulative sum of squares)
framework, without correcting for the mean: the ‘variance’ as inter-
preted in this paper is merely the square of the data value. Since
\[ \text{Var}(X) = E(X^2) - (EX)^2 \]
for any random variable \( X \) possessing
second moments, neglecting to correct for the mean deserves co m-
ment.

In many applications, admittedly, the data can be safely assumed to
have mean close to zero. Examples include returns on many finan-
cial assets (Taylor (2005)); and outputs from wavelet filters, which
involve successive differencing of the data (Gencay et al. (2002, Ch.
4), Percival & Walden (2000, Ch. 4)).

In their seminal paper, Brown et al. (1975) considered CUSUMs
defined on residuals from linear models, and so with zero mean.
The first approach to approximating the distribution of CUSUM
variables seems to be in Hsu (1977), using Edgeworth expansions
and Dirichlet distributions: but she assumed that the mean was a
known constant, to be subtracted from the data before the CUSUM analysis started.

The point is not so clear-cut when the expected value is less well-behaved. Applying a mean correction then typically entails adjusting data for average values over selected intervals, which requires updating whenever another putative breakpoint is considered. An example is the calculation of variance ratios in Ureche-Rangau & Speeg (2011), adapting procedures set out in Tsay (1988), which approach is partly aimed at disentangling level shifts and variance changes.

The gains from neglecting to correct for the mean include simplicity of approach, enhanced intuition and far more straightforward programming. Lavielle (1999) is but one of many writers who emphasise the enhanced facility of modelling when the number and location of breakpoints are known, and one utility of our approach may be to facilitate finding breakpoints, which can act as a springboard for further modelling. Our simulation results seem to indicate that our methods are reasonably accurate for general types of breakpoint. The only preliminary ‘massaging’ required is that it is probably wise to add or subtract a constant in the first instance in order to work with a time series of zero overall mean, which cannot affect the location of change points; there may in addition be the need to ‘perturb’ the data slightly on the odd occasion when the programme identifies two breakpoints during a single sweep of the algorithm, which causes a warning to sound in the mult.loc programme that we are using: this possibility has been ignored in the programming on the basis of its extremely low probability of occurrence in continuous data, but becomes more of a concern to us because we are reflecting data before inputting it into this programme. This issue is more apparent than real, and simple ways around the problem are discussed in §3, and with a practical example dealt with in §5.
2.3 Wavelets

Over the last twenty years wavelets have found many uses in the physical sciences and engineering, and to some extent in the social and medical sciences. With the appearance of Gencay et al. (2002) there was reason to hope for increased application of wavelets to economics and finance, but such has hardly been the case. A recent addition to the literature of wavelets used in economics and finance is the book by In & Kim (2012), which perhaps augurs well for future activity in this area.

In wavelet analysis, linear filters are applied to break down time series data into frequency bands, the mechanics of which boils down to premultiplying a time series vector by large matrices. Roughly speaking, wavelet theory emulates Fourier decomposition of data into frequencies. It is more useful than Fourier decomposition in that the strength of the basis function may alter over time, unlike conventional Fourier analysis, for which the amplitude of a sinusoid remains unchanging. Decomposition of data in wavelet analysis is however not into precise frequencies, but into frequency bands, at different levels: the lowest level or scale (the highest frequency) covers the frequency band \((1/4, 1/2)\), or with ‘period’ or ‘cycle length’ between 2 and 4 (time units); the second level has frequency band \((1/8, 1/4)\), etc. Further details are to be found, \textit{i.a.}, in Gencay et al. (2002, §7.3) and Percival & Walden (2000, Ch. 4)).

As an illustration of the utility of a wavelet decomposition of data, we give two examples. The top graph in Figure 1 on p. 15 shows daily returns on Exxon-Mobil stock over the last 30 years, while that in Figure 2 on p. 17 gives daily returns on Citigroup over the last 10 years. The returns are concatenated: i.e., they are from one daily closing price to the next, regardless of weekends and holidays (so Friday night close to Monday night close for instance provides one return). Prices have been adjusted for stock splits and dividends, and returns are calculated as differences of log prices. Data has been downloaded from Yahoo Finance.

Below the top graph in these Figures appear the decomposition of
Figure 1: 1983-2013 MRA for concatenated returns for Exxon Mobil.
the original signal into components containing the various frequency bands, with the lower wavelet levels (higher frequency bands) appearing towards the top of the diagram. After giving 8 levels of details for these signals, at the very bottom of the Figures appear the 8th smooth, being the residual after subtracting the 8 details from the original signal. Otherwise stated, the bottom 9 graphs sum to the original data plotted in the top graph, although this feature is obscured because the vertical scales in the details vary. This breakdown, the so-called Multi-resolution analysis (MRA), has been calculated using MODWT transforms, and all of these graphs are aligned with the original time axis.

It is the intention of this paper to estimate not just breakpoints for the original time series plotted at the top of these graphs; but also simultaneously to estimate joint breakpoints for the details, appearing as the lower graphs.

The principal wavelet filters are the DWT and MODWT filters. The DWT transform provides independent wavelet coefficient values, but there are fewer and fewer of these as one moves to higher level of wavelet coefficients (lower frequencies); there is then the problem of allocation of a change point for a higher level of DWT coefficient to the original time series. For the alternative MODWT coefficients, the allocation of a given coefficient to the original time values is less difficult, but the coefficients are not independent of each other.

The function testing.hov is designed to apply both of these filters to data and input the results into mult.loc to identify the breakpoints at all scales for both series of transforms. The difficulty lies in identifying the time values in the original data which correspond to the breakpoints produced by mult.loc.

For conventional searching for breakpoints of a single time series, and in the absence of any wavelet decompositions, one merely utilises mult.loc, with suitable dummying of input to feed the same data into both DWT and MODWT lists; the outputs for both DWT and MODWT are then identical, and one can jettison the latter.
Figure 2: 2003-2013 concatenated returns for Citigroup.
3 The CUSUM Method

As a preliminary to estimating breakpoints for more realistic examples, we illustrate the CUSUM methodology on some very simple examples. The \texttt{R} code used is adapted from the first few lines of the \texttt{mult.loc} function reproduced in Appendix B on p. 80.

For a general time series of length \( N \), with transpose given by
\[
X_0^T = (x_1, x_2, \ldots, x_{N-1}, x_N)
\]
and adapting notation from Gencay et al. (2002, p. 247) and Percival & Walden (2000, p. 380), we write
\[
P_k = \frac{\sum_{j=1}^{k} x_j^2}{\sum_{j=1}^{N} x_j^2}
\]  
where \( 1 \leq k \leq N - 1 \); we further define
\[
D^+ = \max_{k} \left( \frac{k}{N - 1} - P_k \right) ; \quad D^- = \max_{k} \left( P_k - \frac{k}{N - 1} \right)
\]
and finally

\[ D = \max (D^+, D^-) \]

Our example starts with \( X^T = (1, 2, 3, 7, 8, 9, 10, 16, 17, 18) \), with clear breakpoints at 3.5 and 7.5. The calculations of \( D^+, D^- \) and \( D \), for \( X \) input directly and then in reverse order, are illustrated in Figure 3 on p. 18. The value of \( D^+ \) is the maximum distance below the red line in those figures, while \( D^- \) is the maximum distance above the green line. The maximum \( D \) of \( D^+ \) and \( D^- \) is shown in those diagrams as occurring at times \( \text{loc.dwt} \), the terminology taken from the function \( \text{mult.loc} \), as illustrated in Table 1 on p. 20. Once \( D \) has been calculated, it is tested for significance, usually at the 5% level, a significant value leading to the claimed discovery of another breakpoint.

The time series examples in Figure 3 on p. 18 have clear breakpoints \( \text{loc.dwt} \) at times 3 and 7, with \( D \) pinpointing the larger break in the series. That is not to say that the value of \( D \) found is significant: the critical value for such a small sample is not known, not least because our methods are model-free.

Asymptotically, the distribution of \( D \) is based on that of the maximum absolute value of a Brownian bridge. Provided that \( N \geq 128 \), the asymptotic distribution of \( D \) is found to be acceptable in practice, according to Whitcher et al. (1998) (see also Gencay et al. (2002); the density is given in Percival & Walden (2000, p. 381) and Billingsley (1968, p. 85)).

Proceeding to the simple example of \( X^T = (1, 2, 3, 4, 5) \), with its forward reflection of \( X_1^T = (1, 2, 3, 4, 5, 4, 3, 2, 1) \), the calculations are illustrated in Figure 4(a) on p. 22. The values of \( D^+ \) utilise the red upper line, from which the \( P \) value is subtracted, so that \( D^+ \) assumes positive values for low values of \( P \); while \( D^- \) values are found from the green lower line, being positive for high \( P \) values. From the symmetry, the maximum \( D^+ \) value found, shown in red, is equal to the maximum \( D^- \) value, shown in green. As expected, then, there are two breakpoints indicated, viz. 3 and 7.

Changing the data to \( X^T = (1, 2, 3, 6, 7) \), with a definite breakpoint
x=c(1:5,5:1);N=length(x)
P <- cumsum(x^2)/sum(x^2)#modelled on first few lines of mult.loc
test.stat <- pmax((1:N)/(N - 1) - P, P - (1:N - 1)/(N - 1))
loc.dwt <- (1:N)[max(test.stat) == test.stat]
#which produces the output
> P
[1] 0.009090909 0.045454545 0.127272727 0.272727273 0.500000000 0.727272727 0.872727273 0.954545455 0.990909091 1.000000000
> test.stat
[1] 0.10202020 0.17676768 0.20606061 0.17171717 0.05555556 0.17171717 0.20606061 0.17676768 0.10202020 0.11111111
> loc.dwt
[1] 3 7
#the plot is shown in Figure \ref{rg610}

####further example
x=c(1:3,6:7,7:6,3:1);N=length(x)
P=cumsum(x^2)/sum(x^2)
test.stat <- pmax((1:N)/(N - 1) - P, P - (1:N - 1)/(N - 1))
loc.dwt <- (1:N)[max(test.stat) == test.stat]
> P
[1] 0.005050505 0.025252525 0.070707071 0.252525253 0.500000000 0.747474747 0.929292929 0.974747475 0.994949495 1.000000000
> test.stat
[1] 0.10606061 0.19696970 0.26262626 0.26262626 0.19191919 0.05555556 0.19191919 0.26262626 0.19696970 0.10606061 0.11111111
> loc.dwt
[1] 7
> test.stat[7]-test.stat[3]
[1] 5.551115e-17

Table 1: R output for double breakpoints illustrated in Figure 4 on p. 22
at time 3.5, leads to $X_1^T = (1, 2, 3, 6, 7, 7, 6, 3, 2, 1)$, and yields Figure 4(b) on p. 22. We again have the red line indicating the maximum value of the $D^+$ vector, and the green line indicating the maximum value of the $D^-$ vector; and the lengths of these lines are identical. The preference for the green line evinced by the computer is based on rounding error: the lengths of the red and green lines differ by the order of $10^{-17}$, as indicated at the bottom of Table 1 on p. 20.

Despite the simplicity of these examples, an important principle emerges. In obtaining breakpoints for reflected data, initially one may find two breakpoints, or conceivably more. The computer may pick up one or the other, or both, of the breakpoints, depending on internal rounding errors.

Although the computer expects but one breakpoint over any interval, and produces a warning when more than a single breakpoint is found, in fact it seems to choose the first breakpoint found and proceed with the search. Estimated breakpoints can still be extricated from the finished computer run, but many errors are listed, and the standard output is not produced.

The problem is not particularly serious, and simple remedies are at hand, the simplest being to perturb the time series values by extremely small random values: this will not impact on sensible choices of breakpoints. Alternatively, especially if one is not using wavelets, one could reflect the data and then drop the first value, say: this would suffice to remove the symmetry causing the potential problem, with little material difference in choice of breakpoints.

4 Searching for breakpoints

Before we proceed to more realistic examples, we pause to discuss the search procedures in `mult.loc` when there are several breakpoints.
An obvious approach to finding change points is to set the first value of the time series to $x_{t_1}$ and the final value to $x_{t_2}$, say, so that initially at least $t_1 = 1, t_2 = N$. One tests between $t_1$ and $t_2$ to see whether there is a change point between $x_{t_1}$ and $x_{t_2}$: if not, then the algorithm is complete. Should there be a change point identified in that range, say $t_3$, we set up a recursion by seeking all breakpoints in the time interval $(t_1, t_3)$, then all breakpoints in the interval $(t_3, t_2)$.

This is essentially the algorithm used in the function `mult.loc`, which differs slightly from that used in Inclan & Tiao (1994). The particular algorithm adopted for finding change points need not concern us; but it is worth noting that varying the search method may in general influence which points are chosen; it will at least influence the order in which breakpoints are found, and may impact on their locations.
Within a particular interval \((t_i, t_j)\), Inclan & Tiao (1994) test whether the maximum CUSUM value \(D\) is above or below a critical value, \(D_0\) say, depending on the interval length \(t_j - t_i\): a value of \(D\) above \(D_0\) is significantly greater than 0, indicating a change point in that interval. Those authors took their examples from uncorrelated normal variates with piecewise constant variance, as in our first non-trivial example below, and used the known distribution of the maximum absolute value of the Brownian bridge for their critical values. As noted previously, the Brownian bridge approximation to the finite distributions of CUSUM is only reliable for sample size exceeding 128, which limit has been imposed in the function \texttt{mult.loc} in the \texttt{R} package \texttt{Waveslim} (Whitcher (2013)).

Also as noted in §2.3 on p. 14, \texttt{mult.loc} accepts DWT and MODWT filtered inputs from its parent function \texttt{testing.hov}. This input assumes the form of two lists, \texttt{dwt.list} and \texttt{modwt.list}, each of which contains time series data, together with the start and finish times identifying that part of the time series to be analysed.

The function \texttt{mult.loc} searches the time series in the \texttt{dwt.list} in the manner adumbrated above, but not the time series in the \texttt{modwt.list}. The search for breakpoints for the latter list simply mimics the search over the former list, in a way to be clarified below.

When estimating breakpoints for wavelet details both filters will play a vital role. In estimating breakpoints for a single time series, however, we use the same data for both lists, and identical breakpoints will be produced, as is clear from the matrix produced by \texttt{mult.loc} shown in Table 2 on p. 29: we then simply jettison the results from the MODWT list.

### 4.2 The ancillary role of MODWT in \texttt{mult.loc}

The role played by the ancillary MODWT time series in \texttt{mult.loc} is an odd one, and a brief explanation is in order. Suppose that the DWT time series is \(X^T = (x_{t_1}, x_{t_1+1}, \ldots, x_{t_2})\), and that in the
MODWT list the time series is $Y^T = (y_{s_1}, y_{s_1+1}, \ldots, y_{s_2})$, where the sample sizes $t_2 - t_1 + 1$ and $s_2 - s_1 + 1$ may differ.

Once the DWT process has tested the interval $(t_1, t_2)$ and decided that the putative breakpoint at $t_3$ is significant, then the MODWT process also calculates $s_3$ between $s_1$ and $s_2$. But the calculations of breakpoint locations proceed independently of each other, each located at the location corresponding to the respective $D$ values, and successive breakpoints may move away from each other. Should the DWT process decree that there is no breakpoint over $(t_i, t_j)$, and to move the search over to $(t_k, t_l)$, the MODWT process also abandons $(s_i, s_j)$ and moves over to $(s_k, s_l)$. But even when the original sample sizes are identical, for a given $j$ there is no guarantee that $t_j$ and $s_j$ are going to be anywhere near each other: the only guarantee is that the ordering of points selected is preserved from one series to the other.

One could conceivably strike a problem with this ‘mimicking’ algorithm when the search over the DWT time series wants to continue, but the MODWT search collapses to just one or two points. In our context this seems unlikely for at least two reasons: searches in DWT over an interval of length less than 128 are rejected out of hand, as noted in §3 on p. 18; and the DWT coefficient vector has length at most one half of the MODWT coefficient vector, and for higher levels much less than that.

That point notwithstanding, this problem seems to have arisen in our analysis of Citigroup daily returns analysed in §7 on p. 53, for which the backwards reflection of the data fed into the function testing.hov produced an error. After identifying a dozen or so breakpoints at the first wavelet level, there developed an error arising from the MODWT side of the programme. It appears that the MODWT search for breakpoints ‘ran out’ of points in some sense: to say more is difficult, because the error produced by the computer was not so informative, but one infers that it was of the above nature. The backwards reflection of the data was perturbed slightly, along the lines indicated previously, but the error remained.
Whatever the reason for the error, the analysis of Citigroup returns in §7 proceeds on the basis of the conventional forward and backward pass of original data, and the forward and backward pass of the forward reflection of the data, as will be evident in a glance at Figure 26 on p. 68, and succeeding graphs.

This odd setup for the algorithm arises because of the behaviour of the two basic wavelet filters. The DWT produces truncated output; but it is the DWT output which is approximately uncorrelated, and hence eligible for testing whether or not the putative breakpoint is significant or not. The MODWT output is autocorrelated, and does not satisfy the conditions needed to test the value of $D$ obtained. On the other hand, the latter filter produces output of the same length as the original data, and it is easier to line up the MODWT breakpoints with the original data. It is desirable to retain both sets of breakpoints in trying to determining where breakpoints really lie for the original time series, as will be seen later.

After all that, when not concerned with wavelet analysis, we feed the same data into both the DWT and MODWT dummy time series; the MODWT process simply mimics that of the DWT, and can be disregarded.

5 An extended example with the normal distribution

We simulate independent zero mean normal variates with piece-wise constant variance, for which we estimate breakpoints by 3 methods:

1. we estimate breakpoints for the original (simulated) data utilising CUSUM, viz. ignoring testing.hov and
   (a) inputting the original time series into mult.loc;
   (b) inputting the reversed data into mult.loc;
   (c) reflecting the data forward and inputting it into mult.loc;
(d) reflecting the data backward and inputting it into `mult.loc`.

For the 3rd and 4th of these methods we expect to obtain roughly twice as many breakpoints as for the first two. Results are gathered together in Figures 5, 6, 7 and 8 on pp. 27, 28, 28 and 31 respectively.

2. we estimate breakpoints for wavelet details by applying CUSUM to the wavelet coefficients:

(a) inputting the data itself into `testing.hov`;
(b) inputting the reversed data into `testing.hov`;
(c) reflecting the data forward and inputting it into `testing.hov`;
(d) reflecting the data backward and inputting it into `testing.hov`.

By these means we obtain breakpoints for all levels of the wavelet transforms, but restrict ourselves here to the 4th level, for which results are shown in Figures 9, 10, 11, 12 and 13 on pp. 42, 42, 43, 43 and 44 respectively.

3. Finally we repeat the first step for the individual wavelet details: looking at Figures 1 on p. 15 and 2 on p. 17, the first step has estimated the breakpoints of the original series at the top of the graphs; now we apply the same procedure with the details, viz. the graphs lower down the page, save for the very last one, which is a smoothed or detrended version of the original data. We restrict ourselves again to the 4th level, the 4th wavelet detail labelled $D4$ in the graphs showing the MRAs, for which results are shown in Figures 14 and 15 on pp. 45 and 45.

5.1 CUSUM for the original time series $z_0$, and its reversal and reflections

Breakpoints are estimated for a simulated sample of independent zero mean normal variates, with variance piece-wise constant in blocks of length 1024.
This particular length is chosen because later on it will be convenient to use this data for illustration of wavelet analysis, for which a dyadic sample size will be convenient \((1024 = 2^{10})\). Standard deviations in the blocks are successively 1, 3, 1, 4, 5 and 1, so that the total length of the time series, labelled as \(zz0\), is 6144. The vector \(zz0\) is plotted as the top graph in Figure 5 on p. 27. The graph is a popular one: the basic time series is further plotted in Figure 6 on p. 28, as well as in Figures 7 on p. 28 and 8 on p. 31.

In order to estimate the breakpoints of \(zz0\) we apply the function \texttt{mult.loc} in \texttt{waveslim}, which is listed in Appendix B on p. 80. One could rewrite the function to cater for our precise needs, but the modifications are sufficiently minor that it seems easier simply to ‘dummy’ the input into the extant function.

As already noted, the function \texttt{mult.loc} is set up to accept two time series from its covering programme \texttt{testing.hov}, viz. the DWT and MODWT coefficients obtained by applying those wavelet filters to the original data. The input into \texttt{mult.loc} accordingly contains two
lists, \textit{dwt.list} and \textit{modwt.list}, each of which contains a vector of coefficients, together with initial and final indices or ‘times’ defining the interval over which the computer is to search for breakpoints.

The function \texttt{mult.loc} is recursive: the ‘Recall’ function lower down in the code stands for the function \texttt{mult.loc} itself, which has the advantage that the function will still work should the original name of the function be altered.

The \texttt{R} code for the initial forward pass of the data is shown in the

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6}
\caption{Normal data, breakpoints estimated by CUSUM, 4 methods}
\end{figure}
Leigh Roberts, Breakpoints, February 6, 2014

This R code is to generate \( zz0 \); and its reversal and reflections

\[
m=1024; \quad \text{zz0} = c(\text{rnorm}(m), 3 \times \text{rnorm}(m), \text{rnorm}(m),
\quad 4 \times \text{rnorm}(m), 5 \times \text{rnorm}(m), \text{rnorm}(m))
\]

\[
\text{zz0} = \text{zz0} - \text{mean(zz0)}; \quad \text{zz1} = \text{rev(zz0)}; \quad \text{zz2} = c(\text{zz0}, \text{zz1}); \quad \text{zz3} = c(\text{zz1}, \text{zz0})
\]

#dummying the input to \text{mult.loc}, setting \text{MODWT} equal to \text{DWT}

\[
\text{dwt.list0} = \text{list(dwt=zz0, left=1, right=length(zz0))}
\]

\[
\text{modwt.list0} = \text{list(modwt=zz0, left=1, right=length(zz0))}
\]

\[
\text{wf} = ""; \text{level} = 0; \text{min.coef} = 128; \text{debug} = T
\]

\[
\text{t0} <- \text{mult.loc(dwt.list0, modwt.list0, wf, level, min.coef, debug)}
\]

##the above code produces the output below

Accepted!#the search from 1 to 6144 produces 3074

Going left; using 1 to 3073 ... Accepted!# producing 2047
Going left; using 1 to 2046 ... Accepted!# producing 1027
Going left; using 1 to 1026 ... Rejected!
Going right; using 1028 to 2046 ... Rejected!
Going right; using 2048 to 3073 ... Rejected!
Going right; using 3075 to 6144 ... Accepted!# producing 5122
Going left; using 3075 to 5121 ... Accepted!# producing 4094
Going left; using 3075 to 4093 ... Rejected!
Going right; using 4095 to 5121 ... Rejected!
Going right; using 5123 to 6144 ... Rejected!

##The above details are produced because debug = T

> \text{t0}

\[
\begin{array}{rrrr}
[1,] & 0.28688681 & 3074 & 3074 \\
[2,] & 0.24471500 & 2047 & 2047 \\
[3,] & 0.39992550 & 1027 & 1027 \\
[4,] & 0.30995600 & 5122 & 5122 \\
[5,] & 0.13104280 & 4094 & 4094 \\
\end{array}
\]

##we tidy the matrix up a bit

\[
\text{t0} = \text{t0}[, -4]; \text{t0}[, 2] = \text{round(t0[, 2], 3)}
\]

\[
\text{t0} = \text{cbind(t0[, 3], length(zz0))}
\]

\[
\text{colnames(t0)} = \text{c("lvl", "crit val", "bkpt", "ppn")}
\]

> \text{t0}

##the critical value serves no purpose in this paper

\[
\begin{array}{rrrr}
\text{lvl} & \text{crit val} & \text{bkpt} & \text{ppn} \\
[1,] & 0.287 & 3074 & 0.5003255 \\
[2,] & 0.245 & 2047 & 0.3331706 \\
[3,] & 0.400 & 1027 & 0.1671549 \\
[4,] & 0.310 & 5122 & 0.8336589 \\
[5,] & 0.131 & 4094 & 0.6663411 \\
\end{array}
\]

Table 2: Breakpoints and proportions for normally distributed \( zz0 \): forward pass
#From the last table, we streamline the process a bit
fun.mult.loc=function(zz){
  dwtlist=list(dwt=zz,left=1,right=length(zz))
  modwtlist=list(modwt=zz,left=1,right=length(zz))
  tt=mult.loc(dwtlist, modwtlist, wf="", level=0, min.coef=128, debug=F)
  tt=tt[,c(2,4)];tt[,2]=round(tt[,2],3)
  tt=cbind(tt,tt[,2]/length(zz))
  colnames(tt)=c("lvl","bkpt","ppn")
  return(tt)}#end of fun.mult.loc

tt0=fun.mult.loc(zz0)
tt11=fun.mult.loc(zz11);tt11[,"ppn"]=1-tt11[,"ppn"]
tt22=fun.mult.loc(zz22);tt22[,"ppn"]=pmin(2*tt22[,"ppn"],2-2*tt22[,"ppn"])
tt33=fun.mult.loc(zz33);tt33[,"ppn"]=pmax(1-2*tt33[,"ppn"],2*tt33[,"ppn"]-1)

###
> tt0;tt11;tt22;tt33

<table>
<thead>
<tr>
<th>lvl</th>
<th>bkpt</th>
<th>ppn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3074</td>
<td>0.5003255</td>
</tr>
<tr>
<td>0</td>
<td>2047</td>
<td>0.3331706</td>
</tr>
<tr>
<td>0</td>
<td>1027</td>
<td>0.1671549</td>
</tr>
<tr>
<td>0</td>
<td>5122</td>
<td>0.8336589</td>
</tr>
<tr>
<td>0</td>
<td>4094</td>
<td>0.6663411</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lvl</th>
<th>bkpt</th>
<th>ppn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3074</td>
<td>0.5003255</td>
</tr>
<tr>
<td>0</td>
<td>2047</td>
<td>0.3331706</td>
</tr>
<tr>
<td>0</td>
<td>1027</td>
<td>0.1671549</td>
</tr>
<tr>
<td>0</td>
<td>5122</td>
<td>0.8336589</td>
</tr>
<tr>
<td>0</td>
<td>4094</td>
<td>0.6663411</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lvl</th>
<th>bkpt</th>
<th>ppn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3074</td>
<td>0.5003255</td>
</tr>
<tr>
<td>0</td>
<td>2047</td>
<td>0.3331706</td>
</tr>
<tr>
<td>0</td>
<td>1027</td>
<td>0.1671549</td>
</tr>
<tr>
<td>0</td>
<td>5122</td>
<td>0.8336589</td>
</tr>
<tr>
<td>0</td>
<td>4094</td>
<td>0.6663411</td>
</tr>
</tbody>
</table>

Table 3: Normal data. breakpoint estimates from 4 basic methods
mat0=fun.matt.new3(tt0,NN=NN0)
mat11=fun.matt.new3(tt11,NN=NN0)
mat22=fun.matt.new3(tt22,NN=NN0)
mat33=fun.matt.new3(tt33,NN=NN0)
mat0;mat11;mat22;mat33

[1,]  1000  167  333  500  666  834
[2,]    0    3    2    1    5    4
[1,]  1000  167  333  500  666  833
[2,]    0    5    4    1    3    2
[1,]  1000  166  167  333  334  500  500  665  666  833  834
[2,]    0    10    3    2    9    1    4    8    6    7    5
[1,]  1000  167  168  333  334  500  500  666  666  833  834
[2,]    0    6    7    5    8    1    2    4   10    3    9
# we combine these results:
mtt=fun.matt.new2(mat0,mat11,mat22,mat33,NN=NN0)

[1,]  166  167  168  333  334  500  665  666  833  834  1000
[2,]    1    4    1    5    1    6    1    5    3    3    4

Table 4: Breakpoint estimates for normal data from 4 basic methods, using R functions given in Appendix A in Table 8 on p. 78.
top of Table 2 on p. 29. The vector \(zz0\) is defined as described previously; and for later use, also defined are its reversal \(zz1\) and the reflections \(zz2\) and \(zz3\): these are all plotted in Figure 5 on p. 27. There follow the definitions of \(dwt.list\) and \(modwt.list\); the definitions of \(wf\) and \(level\) are immaterial, although needing to be input; the parameter \(min.coef\) is 128 in line with the discussion elsewhere in this paper, and setting \(debug\) to ‘T’ for ‘TRUE’ will allow us to follow the logic of the programme as it searches for successive breakpoints. After some tidying up, including inserting my own comments after asterisks, results from inputting the original data \(zz0\) are given in Table 2.

It is convenient to change the format of the output obtained in Table 2, stressing the proportion of the sample at which breakpoints occur, in order to facilitate comparison with other collections of estimated breakpoints. This transformation is effected towards the bottom of Table 2, and streamlined somewhat in the following Table 3 on p. 30. The latter Table also shows the results of the similar procedure for the reversed data \(zz1\) and reflected data \(zz2\) and \(zz3\).

There remains the task of gathering these various collections of putative breakpoints together to consider just what breakpoints it is reasonable to attach to this data. Before doing so we note that the command

\[
tt33 = \text{fun.mult.loc}(zz33)
\]

producing the final part of Table 3, did not work originally. Our procedure ran foul of the same problem as arose for the time series in Figure 4(b) on p. 22. By dint of some labour one can in fact still identify the breakpoints found by examination of the ‘Accepted!’ and ‘Rejected!’ portions of the output produced by setting the debug parameter to be TRUE, but the standard output was not produced. As suggested previously, a slight perturbation of the data allowed matters to proceed. The minimum absolute value of the \(zz0\) values was of the order of \(10^{-4}\), and the perturbation defined by

\[
zz33 = zz33 + \text{rnorm(length}(zz33)) * 10^{-9}
\]

was sufficient to produce the final few rows of Table 3.

From the column labelled ‘breakpoint’ in Table 3 on p. 30 were
produced the graphs in Figure 5 on p. 27; the proportions (labelled ‘ppn’) in the final column of the matrices \( tt_{11}, tt_{22} \) and \( tt_{33} \) have, on the other hand, been transformed to proportions of the original data \( zz_0 \) as follows:

\[
\begin{align*}
\text{for } zz_{11}: & \quad p \rightarrow 1 - p \\
\text{for } zz_{22}: & \quad p \rightarrow \min(2p, 2 - 2p) \quad (2) \\
\text{and} & \\
\text{for } zz_{33}: & \quad p \rightarrow \max(2p - 1, 1 - 2p) \quad (3)
\end{align*}
\]

Using these altered proportions produces the graphs in Figure 6 on p. 28, in which the lower two graphs each contain 10 breakpoints, and the upper two five breakpoints. Occasionally we shall separate out breakpoints according to the parts of (2) and (3) which are operative: when the first term in (2) exceeds the second, for instance, one obtains the breakpoints drawn in Figure 16(c) on p. 48; while when the second term dominates, the breakpoints in Figure 16(d) appear. Similar breakdowns of the origin of breakpoints appear in Figures 19, 30 and 31 on pp. 52, 72 and 73 respectively.

Turning then to Table 4 on p. 31, we see that the first breakpoint found for \( zz_0 \) in Table 3, viz. at proportion 0.5003255, appears as column 4 in the matrix \( mat_0 \). The breakpoint in question is in fact 0.5003255 \times 6144 + 0.5, in line with the discussion in §3 on p. 18, although we disregard the final half given all of the uncertainties in the estimation of breakpoints. This proportion falls within the interval \((500 \pm 0.5)/1000\), accounting for the entry in column 4 of the matrix \( mat_0 \). Likewise the lowest breakpoint indicated in passing \( zz_0 \) through \texttt{fun.mult.loc} is a proportion 0.167 of the way through the sample, and appears in column 2 of \( mat_0 \) as the third breakpoint found.

The further definition of the matrix \( mtt \) in Table 4 on p. 31 gathers together the information in the matrices \( mat_0, mat_{11}, mat_{22} \) and \( mat_{33} \), using functions which are listed in Appendix A on p. 78. The value of \( NN_0 \) has been transferred to the final column, with the number of occurrences of that value in the first row of the matrices \( mat_0, mat_{11}, mat_{22} \) and \( mat_{33} \) naturally being 4, as indicated in the final figure in the second row. The other columns gather together
the values of the breakpoints found, as proportions multiplied by \(NN0 = 1000\), and the number of times that they occur in those four matrices in the second row.

Having now combined that breakpoint data into the matrix \(mtt\) with values in the first row and weights in the second row, one can use cluster analysis to try to gather those columns which in fact belong to the one breakpoint. As well as inputting the rows of \(mtt\) separately, one also needs to specify the number of clusters desired.

Specifying 4 clusters to be identified, the commands
\[
dumclust=4
\]
\[
kMW=kmeansW(mtt[1,-ncol(mtt)],dumclust, 
weight=mtt[2,-ncol(mtt)])#FactoClass in R
\]
\[
kMW0=matrix(0,nrow=2,ncol=dumclust)
for(i in 1:ncol(kMW0)){
kmWO[1,i]=min(mtt[,-ncol(mtt)][1,kMW$cluster==i])
kmWO[2,i]=max(mtt[,-ncol(mtt)][1,kMW$cluster==i])
}
#
plot(zz0,type="l",main="kmeansW")
for(i in 1:dumclust)abline(v=mtt[,-ncol(mtt)][1, 
kMW$cluster==i]/NN0*length(zz0),col=as.character(i+1))
#
for(i0 in 1:dumclust)polygon(c(kMW0[1,i0],kmWO[2,i0],kmWO[2,i0], 
kmWO[1,i0])/NN0*length(zz0),c(0,0,1,1)+16,col=i0+1)#end i0 dum
produced the Figure 7 on p. 28.

Several comments need to be made at this point. Firstly, of course the choice of 4 clusters in the light of the original specification for simulation of the data is hardly sensible: two of the widely separated breakpoints have been forced into a common cluster, as indicated by the bar at the top of the figure joining the lowest and highest values in that cluster. Secondly, the results of the functions \texttt{kmeans} and \texttt{kmeansW} change with each run: unless told otherwise they start the search algorithm from random starting points. Given that the true breakpoints are equidistant, the cluster pairing two of them can be expected to change from run to run, as can the colouring in the figures, which depends on the order in which clusters are identified.
Thirdly, what happens when we get an extended cluster when we are not working with simulated data possessing very definite and known breakpoints? To use the cluster means as the breakpoints is meaningless for a cluster containing but two very distant breakpoints; but to increase the number of clusters may not solve the problem, because the results of these algorithmic searches are not necessarily nested as the number of clusters increases.

In this case setting the number of clusters to be 5 in the first line of the code shown above suffices to produce Figure 8 on p. 31, which certainly seems to produce satisfactory results. Indeed, the outcomes from all four methods are accurate, to judge from Figure 5 on p. 27, as expected: the simulated data has been chosen carefully to be uncorrelated and have obvious breakpoints; and the normal density has thin tails, minimising the probability of competing breakpoints. The situation will be less clear when the normal distribution is replaced by the thicker tailed and long memory examples, in §6 on p. 47.

Measuring the extent of a cluster in which a breakpoint appears in the manner indicated in Figure 7 on p. 28 may give some indications of reliability of a point estimate of that breakpoint. This falls well short of the statistician’s goal of providing a standard error with a point estimate, but may be as much as one can manage in the absence of an assumed model generating the data. An alternative could be calculating the mean and variance of those points gathered together in a single cluster, and showing the extent of a ‘confidence interval’, perhaps calculating the endpoints as one or two standard errors from the mean in either direction. That might be a sensible approach to take when values within the cluster are numerous and well dispersed; but when there are no intermediate points between the end points, a confidence interval of length two standard errors more or less matches the distance from the minimum to the maximum of the clusters.

Before closing this section, it is worth noting that the desirability of increasing $NN_0$ past a certain point is debatable. As noted previously, one cannot start to identify a breakpoint until one has
passed it, and ostensible accuracy of breakpoints may be spurious. For daily data, it may not make sense to try to locate breakpoints more accurately than say within a week or two. It all depends on the nature of the data; and our conclusions are all the less definite because our approach is model-free.

5.2 The normal distribution: breakpoints of 4th wavelet details by CUSUM applied to wavelet coefficients

The function testing.hov applies the DWT and MODWT filters to its input series, and estimates (through the function mult.loc) the breakpoints of the wavelet coefficients found, for each wavelet level.

The series \( z_0, z_{22}, z_{33} \) are input into testing.hov. Our first concern is to map the resulting (DWT and MODWT) breakpoints found onto the original time axis of the data. Then we have the usual problem of transforming results for \( z_{22}, z_{33} \) to apply to the original time frame for \( z_0 \); and the further usual problem of culling a large number of resulting putative breakpoints to retain those which are genuine, and rejecting those which can be discarded or merged.

5.2.1 Translation of wavelet coefficient breakpoints into the original time scale

Suppose a vector \( x \) of length \( N \), \( N \) being a suitable multiple of a dyadic number: if the desired wavelet depth were four, for instance, the number \( N \) would need to be a multiple of 16 = \( 2^4 \) to allow the DWT filtering to proceed.

Let \( L \) be the length of the underlying wavelet filter used (\( L = 8 \) here because we are using the ‘la8’ filter), \( j \) be the depth of the wavelet transform, and \( L_j \) the length of the \( j \)th level filter:

\[
L_j = (2^j - 1)(L - 1) + 1
\]
Setting $\text{loc.dwt}$ and $\text{loc.modwt}$ to be the vectors of estimated breakpoints returned by testing.hov$(x)$ (in accord with the notation used in the function mult.loc, listed in Appendix B on p. 80; also see the code at the top of Table 1 on p. 20), the estimated original time points to which they correspond are given by

\begin{align*}
\text{tm.dwt} &= (\text{loc.dwt} + 1) \times 2^j - 1 - |\nu^H| \\
\text{tm.modwt} &= \text{loc.modwt} - |\nu^H|
\end{align*}

in an obvious notation; and in which

\[ |\nu^H| = \frac{L_j}{2} \]

These results, and basic notation, are taken from Percival & Walden (2000, pp. 96, 155, 198)), and can be elucidated, somewhat intuitively, as follows.

Let $y_t$ denote the result of applying the $j$th level MODWT filter to the original time series $x_t$:

\[ y_t = \sum_{k=0}^{L_j-1} h_{j,k} x_{t-k} \]

Roughly speaking, the MODWT coefficient $y_t$ corresponds on average to the original time $t - L_j/2$, as reflected in the equation (5). The same logic applies to the DWT, save that the length of the DWT coefficient vector is $N \times 2^{-j}$, which accounts for the factor of $2^j$ applied to equation (5) before subtracting half of the filter length.

5.2.2 Application of testing.hov to produce breakpoints for the 4th detail of the MRA of $zz0$

The results of inputting $zz0, zz11, zz22$ and $zz33$ into testing.hov and applying equations (4) and (5) are shown in Figure 9 on p. 42. The underlying R code for these graphs is shown in Table 9 on p. 79. The detailed output for testing.hov($zz0$) is shown in Table 5 on p. 40, with the first output shown there similar in form to that
in Table 2 on p. 29; less detailed output is shown in Table 6 on p. 41 for the cases in which \( zz_{11} \), \( zz_{22} \) and \( zz_{33} \) are input to testing.hov.

The top graph in Figure 9 contains 4 breakpoints from the more reliable DWT method and 4 from the perhaps less reliable MODWT method, corresponding to the 4 lines of output in Table 5 for wavelet level 4. Analogously, there are 4 breakpoints for each method shown in the second graph in Figure 9 and 8 breakpoints in the final graphs, in agreement with the number of breakpoints identified in Table 6. The agreement between the 4 data passes is reasonable, with the discrepant breakpoints on the extreme left of the 3rd graph and on the extreme right of the 4th graph both arising from the MODWT method.

Figures 10, 11, 12 and 13 on pp. 42, 43, 43 and 44 respectively, combine the breakpoints produced by the 4 ways of passing data into the computer, and recast the results as breakpoints arising from the DWT transform; the MODWT transform; and finally for both transforms together. Superimposed on these latter four Figures are the results of a cluster analysis effected on the breakpoints, into 5, 5, 8 and 10 clusters respectively.

Both Figures 10 and 11 were produced by identical computer code, and highlight the variety of results obtainable from cluster analytical methods. As regards the DWT breakpoints in the top graphs, Figure 11 seems to do a much better job in identifying breakpoints than the Figure 10; whereas the MODWT results in the second graphs give widely varying clusters, and seemingly inappropriate gathering of breakpoints into single clusters. The variation in results comes about because we have merely specified the number of clusters in the computer runs of the function \texttt{kmeansW}, leaving the computer to start the searches from randomly chosen points. It is also possible to input the starting values for the algorithm, in which case the output would be deterministic.

Results as variable as these clearly need to be treated with some caution, but allowing the number of clusters to increase to 8 and 10 in Figures 12 and 13, on pp. 43 and 44 respectively, allow of some
mild optimism, in that DWT breakpoints in both of those graphs seem to be allocating breakpoints sensibly over the time series; or at least there is far less obviously inappropriate melding of distant breakpoints into the one cluster which we saw with lower numbers of clusters.

Several points arise. Firstly, how reliable are our estimated breakpoints from the MODWT transform, since the DWT method is the one founded on ‘proper’ statistical techniques? Secondly, how close can breakpoints get before they can be considered to be random appearances of the same breakpoint?

Thirdly, the clustering algorithm includes all identified breakpoints however low the weightings. We have already noted the outliers from the reflected data in Figure 9, in the 3rd graph to the left of the figure, and to the right of the 4th graph. Each of these outliers is unsupported by the other 3 methods, and it seems logical to remove them from consideration: each of these points turns out to be either a cluster by itself, or it extends another cluster hugely.

Clearly one wants to combine the mechanical algorithmic approach of the cluster analysis with visual examination of the breakpoints. The final graphs in Figures 10, 11, 12 and 13 each contain 48 putative breakpoints, and imposing even 10 clusters is clearly unsatisfactory if one believes that the MODWT breakpoints are reliable.

One can take the mathematical side of things a stage further by considering a quasi ANOVA approach based on the sums of squares within and between clusters as the number of clusters changes. As pointed out in Kendall (1965, p. 39), the ratios for this ‘F test’ do not in fact have the $F$ distribution, but the approach may still assist in deciding on how many breakpoints we wish to retain. We do not however adopt this approach in this paper.

Further possibilities include generalising the weights of the putative breakpoints. One could regard each breakpoint as generating weights on either side, say triangular or exponentially declining, the idea being that finding breakpoints by CUSUM is sufficiently loose
Table 5: Full results from testing.hov(zz=0)

that identifying a breakpoint merely indicates a general region in which the ‘real’ breakpoint lies. Another possibility is to allow different weights assigned to a cluster to be reflected by some means in the graphs plotted of the breakpoints, perhaps by varying the height of the bars in Figures 10, 11, 12 and 13 to reflect the total weights of the clusters in question. Again, we make no attempt to do this in this paper.
Table 6: Partial results from testing.hov (zz11), (zz22) and (zz33)
Figure 9: Normal data, $zz_0$D4: breakpoints from testing.hov($zz_0$), testing.hov($zz_11$) etc., from Tables 5 and 6

Figure 10: Normal data, $zz_0$D4: reconciliation of DWT breakpoints from Figure 9; then MODWT, then both; 5 clusters
Figure 11: Normal data, $zz_0$D4: reconciliation of DWT breakpoints from Figure 9; then MODWT, then both; 5 clusters

Figure 12: Normal data, $zz_0$D4: reconciliation of DWT breakpoints from Figure 9; then MODWT, then both; 8 clusters
5.3 The normal distribution: direct CUSUM for the 4th MRA detail

Finally we turn to direct CUSUM estimation of breakpoints for the 4th detail $z_0 D_4$ of the MRA for $z_0$, the third method listed in the preamble to §5 on p. 25. We fit breakpoints by applying CUSUM forwards and backwards, for the original data and for the reflected data, as previously done for $z_0$ itself. While Figure 14 on p. 45 reminds us of the comparative behaviour of the underlying time series $z_0$ and its fourth MRA detail, the breakpoints are shown in Figure 15 on p. 45.

At first blush it is encouraging to note that all 4 methods seem to produce breakpoints which are consistent with each other. Less satisfactory however is the very large number of breakpoints produced: the contrast between Figure 9 on p. 42 and Figure 15 could hardly be greater. Consider for instance the first 1000 points, with multiple breakpoints indicated in Figure 15, compared with a single
Figure 14: Normal data zz0 and the 4th MRA detail zz0$D4$

Figure 15: Normal data, 4th MRA detail: direct estimation of breakpoints from CUSUM, 4 methods
breakpoint in Figure 9.

The breakpoints identified over those first 1000 points in Figure 15, while they certainly reflect locally significant irregularities in the D4 series, correspond to relatively small changes in value in comparison with the underlying time series. A comparison between the two graphs in Figure 14, allowing for the change in scale, indicates that the 4th detail in the second graph is not likely to contain too many breakpoints over the first 1000 points when the scale at which we operate is dictated by the original data. Extending this notion to the entire time series, the original time series varies from -20 to +15, whereas the 4th detail varies from -3 to 3. The CUSUM algorithm will pick up discontinuities in D4 which are not necessarily important at the scale of the original data.

A further reason for disregarding some of the breakpoints identified at the 4th detail is that, at least in economics and finance, one does not really expect more breakpoints at the 4th wavelet detail than for the original time series, because to have a breakpoint at a low frequency with smooth coefficients at higher frequencies, and a smooth original time series, would be unusual: a shock to the financial system would normally impact at higher frequencies, and would have less impact at the lower frequencies. One advantage of keeping with the use of CUSUM on wavelet coefficients to estimate breakpoints of wavelet details is that the number of breakpoints is likely to be kept to a manageable value: for this simulated data, the number of breakpoints identified for D4 is four for each method in Figure 9 on p. 42, compared with 5 originally for the original series, as seen in Figure 5 on p. 27. The brake on the number of breakpoints found by the DWT transform is of course fundamentally due to the fact that the number of DWT coefficients falls by a factor of 2 for each unit increase in the wavelet level.

On the basis of the results shown in Figure 15 we decide to estimate breakpoints for wavelet details by applying CUSUM to the wavelet coefficients rather than to the wavelet details themselves. In other words, we shall use the second method rather than the third listed in the preamble to §5 on p. 25. Even then, with our ‘forwards and
backwards’ methods of estimation of breakpoints, we shall have a difficult enough time in isolating the ‘real’ breakpoints.

6 Examples with heavy-tailed and long memory distributions

The simulation of the heavier tailed $t$ distribution in place of the normal distribution does not really produce major surprises, although fewer breakpoints are perhaps identified than one would have expected. The story changes markedly however when we simulate a long memory process, with very few breakpoints identified.

6.1 Simulation of the $t$ distribution with 4 degrees of freedom

For a less well behaved example, we retain the idea of independent drawings of random variables with variance piecewise constant in blocks of length 1024; but this time we draw from the $t$ distribution with four degrees of freedom (df), again multiplied by 1, 3, 1, 4, 5 and 1 successively, mimicking our approach above with the normal distribution. The data is plotted in Figure 16 on p. 48. As indicated by the colour coding of the breakpoints, subgraphs (a) and (b) correspond to the forward and reverse passes of the data respectively; while subgraphs (c) and (d) correspond to the forward and reverse passes of the forward reflection of the data, and (e) and (f) to the forward and reverse passes of the backward reflection of the data. The setup of this graph is discussed briefly immediately following equations (2) and (3) on p. 33.

The forward symmetrisation of the data was again subject to the artificial computer error of finding two breakpoints when only one was expected. The minimum absolute value of the data was of the order of $10^{-4}$, so a perturbation of the order of $10^{-9}$ was effected:

$$zz22=zz22+rnorm(length(zz22))*10^{-9}$$
Figure 16: Breakpoints for simulated $t$ random variates, df=4
with $z_{22}$ now reinterpreted as defined from realisations of the $t$ distribution rather than the normal distribution.

Running symmetrised data forwards and backwards through the programme produces almost the same estimated breakpoints, as indicated in Figures 16(c) and (d) on the one hand, and Figures 16(e) and (f) on the other. The vertical lines indicating breakpoints are however not quite at identical points: consider for example the two breakpoints shown very close together just after index 1000 in Figures 16(c) and (d). In Figure 16(c), breakpoints are shown at proportions 16.5% and 16.8% of the sample value; whereas the corresponding breakpoints in Figures 16(d) are drawn at sample proportions of 16.4% and 16.8%. Assuming that the breakpoints at proportions 16.4% and 16.5% are the same, the distance between the two breakpoints indicated is about 20 time units. Supposing daily data, we have about 3 weeks separating the breakpoints, or a month say if the data has been concatenated over weekends. For financial returns, it may be reasonable to say that there could be two shocks to the financial system in a month; for volatility, on the other hand, with highly autocorrelated and/or noisy data, one may be hard pressed to separate breakpoints so close to each other with any accuracy.

Note also the usefulness of running symmetrised data through the computer. The two final breakpoints, at proportions 91.9% and 94.0%, are firmly indicated in the computer passes over the original data and the backward reflection, but are missed entirely by running the forward symmetrisation of the data through the computer. The first 3 or 4 breakpoints on the other hand are missed completely by looking only at the backwards symmetrisation; but their presence is confirmed by the other runs.

In Figure 17 on p. 50 is shown the fourth detail of the MRA of the simulated $t$ random variables, with 4 df, along with breakpoints fitted by applying CUSUM to the wavelet coefficients via the function `testing.hov`. The DWT and MODWT breakpoints are shown separately in Figure 18 on p. 50, together with both types of breakpoints combined in the third graph. Once again we see that the
Figure 17: DWT and MODWT breakpoints from the 4 approaches for the t4 data, for the 4th MRA detail

Figure 18: Breakpoints from the t4 data, 4th MRA detail; for DWT, MODWT, combined; 10 clusters
DWT breakpoints are providing sharper values of breakpoints compared with the MODWT estimates and the combined estimates. The MODWT analysis indicates breakpoints at indices of about 4200 and 4300, which do not appear in the DWT analysis. These would appear to be genuine breakpoints for the 4th MRA detail, and are presumably separate breakpoints despite being included in a common cluster (the green coloured bar) in Figure 18(c).

6.2 Simulation of a long memory process: the Hosking approach

Our final simulation is for a long memory process, generated by the function `hosking.sim` in waveslim (Whitcher (2013, p. 36)), and originally based on Hosking (1984). The sample size is 2048. As for the last example, the forward reflection of the data needed a slight nudge to persuade the programme to work its magic, and the results are given in Figure 19 on p. 52, and Table 7 on p. 53.

The utility of running symmetrised reflections of the data forward and backward is revealed. From Table 7 the 13th and 9th breakpoints found from the reverse run of the data reflected forward (in the 3rd bottom line) are not picked up by any other method, whereas other breakpoints, as well as their location, seem to be consistently picked up by all of the methods. Whether to choose a breakpoint at about the 85% point of the sample on the basis of its being picked up from one method out of 6 is by no means clear; and the same story is told in Figure 19(d) on p. 52.

When it comes to applying the function `testing.hov` to the four variations of the data input, we obtain an odd result. The original Hosking data and its reverse are input into `testing.hov` with sensible output; but no breakpoints are found, at any wavelet level, for the reflected data. We obtain, in the notation from Table 9 on p. 79 in Appendix A, and noting that \(zz\{0\} is now redefined as the Hosking data:
Figure 19: Breakpoints for $zz_0$ using the Hosking distribution
the final two lines of which are indicated by the breakpoints drawn in Figure 21 on p. 55; but the errors produced in trying to calculate \( \texttt{ttt}22 \) and \( \texttt{ttt}33 \) boil down to saying that the computer found no breakpoints at all for the reflected data.

Some sense can be made of this by looking at the DWT wavelet coefficients in Figure 20 on p. 54. The breakpoints identified in the vectors \( \texttt{ttt}0 \) and \( \texttt{ttt}11 \) listed above look sensible in the light of the second and third graphs in Figure 20; and it further seems apparent that picking out breakpoints for such wildly fluctuating data would not be easy: there are relatively few features to make one point stand out from its neighbours. Nor is the situation alleviated when the length of the series is doubled: all that is happening is that the denominator in expression (1) on p. 18 doubles, making it even harder to distinguish breakpoints.

\begin{verbatim}
1 2 3 4 5 6 7 8 9 10 11 12
1 100 39 44 51 55 65 68 78 84 86 89 90
2 2048 2 4 3 1 7 6 5 8
3 0
4 2048
5 11 6 8 7 1 5 4 2 3
6 2048 3 6 5 4 8 7 2 10
7 22 15 17 16 1 14 11 12 13 9
8 2048 10 12 11 6 15 14 13 16
9 33 7 9 8 1 5 4 2 3
\end{verbatim}

Table 7: Breakpoints for \( \texttt{zz}0 \) using the Hosking distribution, with \( NN0 = 100 \)

7 Citigroup returns over the last 30 years

In this section we seek to identify breakpoints in the daily returns on Citigroup over the last 30 years, along with those for the first 4
Figure 20: Hosking data, and DWT wavelet coefficients at levels 1, 2, 3 and 4
Figure 21: Hosking data, 2nd and 3rd details of MRA; breakpoints from reversed run, DWT (red), MODWT (green)
Figure 22: 1983-2013 daily prices for Citigroup (first two graphs), Exxon Mobil (middle two graphs), Verizon (final two graphs).
Figure 23: 1983-2013 daily returns for Citigroup, Exxon Mobil, Verizon.
details of its MRA.

We set the scene by comparing three well-known stocks in widely differing industries over the last 30 years: Citigroup (C, previously CN, in the financial sector, the symbol denoting the tick symbol on the New York Stock Exchange (NYSE)); Exxon Mobil (XOM, in the oil industry); and Verizon (VZ, a telecommunications group). All three are very large corporations, and heavily traded on the NYSE.

Singling out Citigroup, a short history is adumbrated before we investigate breakpoints for Citigroup returns over this period.

### 7.1 Comparison of three stocks

Despite the confusing labels to the graphs, Figure 22 shows daily closing prices on our three chosen stocks: Citigroup (first two graphs), Exxon Mobil (middle two graphs) and Verizon (final two graphs), from December 1983 to July 2013 inclusive.

The price data is concatenated over weekends and holidays; and for each company the upper graph shows the actual stock prices, while the lower graph shows the prices adjusted for splits and dividends.

Note the sharp drop in the Citigroup price from the GFC over 2007 and 2008, and the marked fall in price of Verizon around the time of the NASDAQ collapse in late 1999 to early 2001. While Exxon Mobil was largely insulated from strong stock price movements over those two episodes, their share price fell sharply in the October 1987 crash, as did the prices of the other two companies.

Turning to the returns in Figure 23, the data is set out in the same format, save that now we have graphed prices adjusted for splits and dividends in the upper graph of each pair, and the return calculated from those prices in the lower graph. Daily closing prices have again been simply concatenated.
From the 4th and 6th graphs in Figure 23, returns on Exxon Mobil and Verizon have pretty much the same scaling on the vertical axis, and one can see that the volatility on Verizon is somewhat greater than the volatility on Exxon Mobil, for say the decade from say 1995 to 2004; before and after that decade, the returns on the two companies’ stocks are more or less comparable, and with similar volatilities. Taking cognisance of the altered scale for the returns on Citigroup in the second graph shows that Citigroup stock returns have been more volatile than Exxon Mobil and Verizon over the whole 30 years, as expected for a company pivotal in the financial sector; and the volatility of Citigroup is especially high from late 2007 to late 2009.

7.2 Citigroup

7.2.1 History

To label the history of Citigroup as colourful would be a substantial understatement. The length of its history, with roots traced directly back to 1812, when the City Bank of New York was incorporated in New York State, and with links to an even earlier bank, is not so much the point. It is rather that over most of the intervening two centuries Citigroup has been highly successful and innovative, and has impacted hugely on the worlds of finance, commerce and industry, in America and elsewhere. Over the last decade its star has dimmed substantially, but ostensibly at least it seems to have bounced back from bankruptcy in the wake of the global financial crisis (GFC), and the associated government bailout, and is currently the second biggest American bank.

Over those two centuries the name Citi/City (City Bank, Citibank, Citicorp, Citigroup) has pretty well remained in the title of the bank, with the consistent labelling in part perhaps helping to account for the bank’s presence in the markets over much of that interval. To some extent the ubiquitous upside-down red umbrella logo, used by Citigroup from 1998 to 2007, is just as easily associated with Citi-
group; but not perhaps so positively given the troubles of Citigroup in the first decade of the 21st century.

It is not our purpose to delve into Citigroup’s history more deeply than is needed to throw some light onto the behaviour of its stock prices and returns over the last 30 years, but it is worth highlighting some aspects. We refer throughout to ‘Citigroup’, on the understanding that the actual name changed from time to time.

Several important innovations have been associated with Citigroup. It was one of the first companies to take advantage of the telegraph connection across the Atlantic, since its chairman also happened to be a director of the company laying the first transatlantic cable. Then, immediately following the American civil war, Citigroup moved to a federal charter, from its state charter granted by New York State in 1812, allowing it to assume more government business. As one of the biggest American banks (and the largest by the end of the 19th century), Citigroup was in a strong position to issue US currency and US government bonds, well before the founding of the Federal Reserve in 1913.

Nor did the innovations end there. Citigroup was the first bank to offer travellers cheques; the first bank to offer compound interest on deposits; and the first bank to issue negotiable certificates of deposit. It was also strongly innovative in using technology to set up ATMs ahead of other banks.

Ironically, given their troubles from the 1980s, and especially after the merger with Travellers in 1998, until the latter part of the 20th century Citigroup was known for its competent risk management and solid lending practices. This reputation was in tatters by 2008, with a succession of failures: large losses from overseas lending recognised belatedly in 1987; large real estate losses in the early 1990s, partly in the wake of the savings and loan crisis; the dysfunctional marriage between Citigroup, the largest American bank at the time, and Travellers, a very large financial services firm (one of the largest property/casualty insurers in America, and with many other interests) in 1998; the collapse of that marriage from 2002 on;
consistent and heavy fines for various types of malfeasance; and the heavy investment in collateralised debt obligations throughout the first decade of this century, which sank in value with the onset of the GFC in 2007/8. Citigroup were bankrupt in 2008, but rescued by the American government as being too big to fail. The share price was almost zero from early 2009 to early 2011: after Citigroup had repaid the government bailout by year end 2010, they had a reverse stock split, as easily seen in the graphs of the actual stock prices and adjusted stock prices in the top two graphs of Figure 22 on p. 56.

Still looking at those two graphs, the final part of the prices data, from early 2011 to the end of the series, is in fact identical between the two series, save for a rather dramatic rescaling. As noted, the reason for the sharp increase in share price in early 2011 was a reverse stock split in May of that year, following on from the repayment of government aid to Citigroup: 10 old shares overnight became one new share, with a price increasing from less than $5 to about $45. Increasing the adjusted share price from say 5 to 45 in early 2011 in the top graph to preserve parity before and after the stock split means that the top graph jumps upwards by a large factor. The share price in early 2007, say $60, assumes a value of about $600 in the graph of the adjusted prices.

A dramatic fall in the share price in March 1987 was also a stock split. The additional fall in later 1987 is due to the 1987 crash, which is naturally retained in the adjusted price series. There seem to be further stock splits in early 1999 and late 2000, when sharp falls in the share price fail to be reflected in the adjusted price series.

More details of the history of Citigroup can be found in many sources, in particular from Wikipedia (www.wikipedia.org).
7.2.2 Citigroup daily returns and its MRA

Citigroup daily returns in the second graph of Figure 23 on p. 57 are duplicated as the top graph of Figure 24 on p. 64, with the remainder of the latter Figure containing the first four MRA details. The whole of Figure 24 is in turn replicated in Figure 25, with estimated breakpoints superimposed on all of the graphs; but that belongs to the next part of the story.

A glance at the top of Figure 24 is naturally drawn to the flurry of activity towards the right of the graph around the time of the GFC, but there were also several other situations of enhanced volatility over the 30 years. The reverse stock split in 2011 caused a sharp but short lived response: there is a strong response in the first detail of the MRA (the second graph), which corresponds to a period (in the technical sense of the word) of 2-4 days; by the second detail, this shock seems to have died down, or it is perhaps just that the activity has been swallowed up in the aftermath of the GFC.

The shock at the end of 1988 (of unknown origin) and that in 1998 (presumably the Travellers merger) also exhibit relatively short persistence, being fairly strong at the first MRA level (a period of 2-4 days), but largely dissipating by level two (a period of 4-8 days). The shock in October 1987, evident in the first graph, is still reasonably strong by level two, indicating a rather longer impact on returns, say 2-8 days, or say two weeks (‘days’ are business days).

Roughly speaking then, the shocks in 2011, 1998 and the end of 1988, evident in the top graph in Figure 24, seem to have impacted within say a week, but not much beyond that time frame. The October 1987 crash seems to have had a longer lasting impact on returns, retaining some influence for 1.5 or 2 weeks. The graphs showing the MRA details are misleading, because of the reducing scale as one moves down the Figure, but the persistence of the impact of the GFC is clear: Citigroup was one of the major casualties in the fallout from the GFC, as is apparent from the impact on the original data persisting at least as far as the first 4 levels, corresponding to a period of 2-32 days, or say 6 weeks. The impact of the price dropping
practically to zero in early 2009 is particularly long-lasting.

7.2.3 Citigroup breakpoints

As noted, Figure 24 of the Citigroup returns and its MRA is replicated in Figure 25, with estimated breakpoints superimposed on all of the graphs. Breakpoints of the underlying returns data are also shown in Figure 30 on p. 72, broken down by method of estimation of the breakpoints, along the lines of Figures 16 and 19 on pp. 48 and 52 respectively: these breakpoints have been gathered together in the top graph of Figure 25.

The method used to identify these breakpoints is the second method listed in §5 on p. 25, viz. applying the CUSUM method to wavelet coefficients through the testing.hov programme.

Figure 26 on p. 68 shows the identification of breakpoints for the first MRA detail of the Citigroup returns from the various methods. The first graph shows breakpoints found from a forward pass of the data, with red indicating the DWT points and the green the MODWT points; the second graph gives the breakpoints found from the reverse pass of the data, with the dark blue indicating the DWT points and the light blue the MODWT points; and the third graph shows the breakpoints found from passing the forward reflection of the data, with pink indicating the DWT points and yellow the MODWT points. Passing the backwards reflected data through the computer did not work, which seems likely to be due to problems with the MODWT process ‘running out of room’ when it runs in parallel with the DWT process, on which we commented briefly in §4.2 on p. 23. The breakpoints from the 3 methods which did work are gathered together in the second graph of Figure 25 on p. 65.

The approach for the first detail, labelled as ‘D1’, in Figure 26 is mirrored for the second, third and fourth details in Figures 27, 28 and 29 respectively. Breakpoints identified for detail 2 in Figure 27, for instance, are gathered together in the third graph in Figure 25.
Figure 24: Citigroup 30 year returns, and 4 details of the MRA
Figure 25: Citigroup returns, and details, with breakpoints
Moving from the first to the fourth details in Figure 26 and its succeeding Figures, one first notes that the numbers of breakpoints identified falls dramatically as we move to the lower levels, i.e. as we move to the lower frequency bands. The second point to note is a partial tendency for DWT breakpoints and MODWT breakpoints to be separate. This is most obvious when inputting forward reflected data, in the first detail D1, in the third graph in Figure 26. Of the left hand breakpoints, from 1984 to 1986, all but one arise from the MODWT methodology, as do all breakpoints identified between 2004 and 2006 inclusive; all breakpoints arising from 2010 onwards, on the other hand, are produced by the DWT transform. The effect is less pronounced elsewhere, but again in the third graph in Figure 28, the first few breakpoints are all produced by MODWT methodology.

Further comparing the third graph in Figure 26 with the previous two graphs, the MODWT breakpoints produced from the forward reflected data for the first wavelet level are moreover largely unsupported by breakpoints found from the forward and reverse inputs of the basic data; but this feature seems to disappear for the higher wavelet levels, partly perhaps because there are fewer breakpoints for those levels. In any case, breakpoints produced from backward reflected data would have been a useful point of comparison.

Turning to the overall identification of breakpoints in Figure 25 on p. 65, one again notes the falling numbers of breakpoints as one proceeds down the page; and the breakpoints identified seem sensible, both in relation to the irregularities in their own graphs, and in relation to the underlying data.

While our results seem sensible enough at this aggregate level, there is clearly more work to be done in this area. It would be a simple matter to repeat the analysis for say the last 10 or 15 years worth of data; and one could compare these results with other companies’ returns as well as those on the stock indices, etc. One also ultimately needs to consider the detailed breakpoints against the data, in the light of how many breakpoints one expects to be present. But further analysis lies beyond the scope of this paper.
7.2.4 CUSUM applied directly to the wavelet details

Finally, we recall the third method mentioned in §5 on p. 25, viz. estimating breakpoints for wavelet details, but simply treating those details as time series in their own right. The final two graphs are Figures 30 and 31 on pp. 72 and 73, the first obtaining breakpoints for the original data (and hence also gathered together in the first graph of Figure 25 on p. 65); and the second applying the same techniques (using the function `mult.loc`, but not `testing.hov`) to the 4th MRA detail D4. Using this method there are even more breakpoints for the D4 series than for the original data, and our decision to fit breakpoints to wavelet details through CUSUM applied to wavelet coefficients seems validated.

8 Conclusion

A volume of collected works of Frank Redington, the actuary who initiated the theory of immunisation in finance theory (Redington (1952)), was entitled a ‘Ramble through the Actuarial Countryside’ (Chamberlin (1986)).

While certainly not claiming that the present paper has anything like the intellectual merit and lucid style of Redington’s work, it does indeed feel like a ramble. What started out as a simple means of enhancing estimates of breakpoints by running data forwards and backwards through CUSUM programmes, turned into a mission to obtain breakpoints for MRAs, and wavelet transforms of data more generally. As an example of the methodology we finished up by estimating breakpoints for Citigroup returns. The progression of ideas had its logic; but the paper has turned into something not unlike a ramble.

At the end of it all, one must admit that the exercise has not been totally convincing. The use of cluster analysis, although producing graphs which occupied several pages of the finished product, was
Figure 26: Citigroup returns, first detail of MRA, breakpoints from three methods
Figure 27: Citigroup returns, second detail of MRA, breakpoints from three methods
Figure 28: Citigroup returns, third detail of MRA, breakpoints from three methods
Figure 29: Citigroup returns, fourth detail of MRA, breakpoints from three methods
Figure 30: Citigroup returns, breakpoints found from forward and reverse passes of original and reflected data
Figure 31: Citigroup returns, 4th MRA detail, breakpoints found from forward and reverse passes of original and reflected data
not used in the final example, which was left rather incomplete. Nor has the exercise been totally successful in a technical sense, in that passing reflected data through the programme did not work for the highly irregular Hosking data, and the programme did not complete its job for one of the reflected data series for the Citigroup returns. With the benefit of hindsight, one might have rewritten the code especially for our purposes; but there are on the other hand several advantages in keeping with code that is well established and presents sensible results under widely varying conditions.

Moreover, despite wandering far and wide, many questions remain unanswered. Once breakpoints have been chosen, what does one do? One could fit models of various sorts between the selected breakpoints, or fit jump models with jumps at those breakpoints, etc. The paper has adopted a rather hesitant tone, in that emphasis has been placed throughout on the lack of precision of the number and location of breakpoints, and the rather incomplete nature of using cluster analysis to identify breakpoints which should be merged or otherwise. But in the absence of a model, it seems difficult to be more precise.

The above caveats notwithstanding, we believe that the approach to identifying breakpoints adumbrated in this paper has much to offer finance practitioners. We have noted above that wavelets have so far found little application in finance theory or practice; the consistent identification of breakpoints over various levels of MRA details is an exciting prospect, and offers the finance practitioner opportunities not only better to model the underlying market processes but also their component series at various frequency bands. Given the importance of identifying punters who operate in the financial markets with different horizons, sensible modelling of their behaviours means that sensible and consistent identification of breakpoints at different frequency bands is of some importance. There are other possible uses in finance too: more closely identifying the time at which external shocks impact on stock prices, identifying times at which punters artificially flood markets with proposed trades but do not execute (so-called ‘robot trades’, see for instance Madrigal (2010)), in order to influence price, etc. Wavelets have been much used in fields other than finance; and more accurate identification
of breakpoints for MRA components can also be expected to have benefits in those more general fields of application.

References


Appendix A

Table 8 contains R functions used in the paper, and Table 9 contains R code used for inputting testing.hov for finding breakpoints for wavelet coefficients (applying methods 1 and 2 respectively in the list in §5 on p. 25).

```
fun.matt.new3=function(tt,NN=NN0,eps=1/2){
  mat=as.matrix(numeric(2),nrow=2)
  for(i in 0:NN){#begin i do loop
    w=which(abs(tt[,3]-x0)<=eps/NN)
    if(length(w)==0)next else{
      if(length(w)==1){mat=cbind(mat,c(i,w));next} else{
        mat=cbind(mat,matrix(c(rep(i,length(w)),w),byrow=T,nrow=2))
      }
    }
  }#end i do loop
  mat[1,1]=NN
  return(mat)}#end fun.matt.new3

fun.matt.new5=function(tt,NN=NN0,eps=1/2){
  mat=as.matrix(numeric(2),nrow=2)
  for(i in 0:NN){#begin i do loop
    w=which(abs(tt[,5]-x0)<=eps/NN)
    if(length(w)==0)next else{
      if(length(w)==1){mat=cbind(mat,c(i,w));next} else{
        mat=cbind(mat,matrix(c(rep(i,length(w)),w),byrow=T,nrow=2))
      }
    }
  }#end i do loop
  mat[1,1]=NN
  return(mat)}#end fun.matt.new5

fun.matt.new2=function(mat0,mat1,mat2,mat3,NN=NN0){
  ctr=numeric(NN)
  for(i in 1:NN){
    ctr[i]=ctr[i]+length(which(mat0[1,]==i))+length(which(mat1[1,]==i))+
    length(which(mat2[1,]==i))+length(which(mat3[1,]==i)))
  }
  matt=matrix(c(which(ctr!=0),ctr[ctr!=0]),nrow=2,byrow=T)
  return(matt)}#end fun.matt.new2

fun.matt.new1=function(mat0,mat1,mat2,mat3,mat4,mat5,mat6,mat7,NN=NN0){
  ctr=numeric(NN)
  for(i in 1:NN){
    ctr[i]=ctr[i]+length(which(mat0[1,]==i))+length(which(mat1[1,]==i))+
    length(which(mat2[1,]==i))+length(which(mat3[1,]==i))+
    length(which(mat4[1,]==i))+length(which(mat5[1,]==i))+
    length(which(mat6[1,]==i))+length(which(mat7[1,]==i)))
  }
  matt=matrix(c(which(ctr!=0),ctr[ctr!=0]),nrow=2,byrow=T)
  return(matt)}#end fun.matt.new1
```

Table 8: R function definitions for method 1 in §5
nu=function(L=8)if(floor(L/4)==ceiling(L/4))return(-L/2+1)else if(L==10|L==18)return(-L/2)else if(L==14)return(-L/2+2)else print('what is L; or nu?')
Lj=function(j,L=8)(2^j-1)*(L-1)+1#from PW 96; previously I had divided this by 2
modd.nu.h=function(j,L=8)Lj(j,L)/2+nu(L)-1#PW 156
#b.offset.dwt=function(levl){if(levl<3)return(2*levl+1) else return(6)}
#b.offset.modwt=function(levl){Lj(levl)-1}
J=4#zzrdpa for wavelets for CN
#given zz0=zz0-mean(zz0)zz11=rev(zz0);zz22=c(zz0,zz11);zz33=c(zz11,zz0)
ttt0=testing.hov(zz0,wf="la8",j=3,debug=F)
#zz22=zz22+norm(length(zz22))*10^(-9)
ttt22=testing.hov(zz22,wf="la8",j=3,debug=F)
#zz33=zz33+norm(length(zz33))*10^(-9)
ttt33=testing.hov(zz33,wf="la8",j=3,debug=F)
#
fun.ttt.transf=function(ttt,zz){
 ttt=ttt[,2]
 ttt=cbind(ttt[,1:2],0,0,ttt[,3],0,0)
 colnames(ttt)[c(1,3:7)]=c("wlvl","tmdwt","pndwt","loc.modwt","tmmodwt","pnmodwt")
 ttt[,"tmmodwt"]=ttt[,"loc.modwt"]-modd.nu.h(ttt[,"wlvl"])
 ttt[,"tm0modwt"]=ttt[,"loc.modwt"]-modd.nu.h(ttt[,"wlvl"])
 ttt[,"tm0dwt"]=(ttt[,"loc.dwt"]+1)*2^ttt[,"wlvl"]-1-modd.nu.h(ttt[,"wlvl"])
 return(ttt)}
# ttw0=fun.ttt.transf(ttt0,zz0)
 ttw1=1-2*ttw0[,c("pndwt","pnmodwt")]*length(zz0)
 ttw2=ttw2[,c("pndwt","pnmodwt")]*length(zz0)
 ttw3=ttw3[,c("pndwt","pnmodwt")]*length(zz0)
 ttw4=ttw4[,c("pndwt","pnmodwt")]*length(zz0)
 ttw5=ttw5[,c("pndwt","pnmodwt")]*length(zz0)
 ttw6=ttw6[,c("pndwt","pnmodwt")]*length(zz0)
 ttw7=ttw7[,c("pndwt","pnmodwt")]*length(zz0)
 ttw8=ttw8[,c("pndwt","pnmodwt")]*length(zz0)
 ttw9=ttw9[,c("pndwt","pnmodwt")]*length(zz0)
 ttw10=ttw10[,c("pndwt","pnmodwt")]*length(zz0)
 ttw11=ttw11[,c("pndwt","pnmodwt")]*length(zz0)
 ttw12=ttw12[,c("pndwt","pnmodwt")]*length(zz0)
 ttw13=ttw13[,c("pndwt","pnmodwt")]*length(zz0)
 ttw14=ttw14[,c("pndwt","pnmodwt")]*length(zz0)
 ttw15=ttw15[,c("pndwt","pnmodwt")]*length(zz0)
 ttw16=ttw16[,c("pndwt","pnmodwt")]*length(zz0)
 ttw17=ttw17[,c("pndwt","pnmodwt")]*length(zz0)
 ttw18=ttw18[,c("pndwt","pnmodwt")]*length(zz0)
 ttw19=ttw19[,c("pndwt","pnmodwt")]*length(zz0)
 ttw20=ttw20[,c("pndwt","pnmodwt")]*length(zz0)
 ttw21=ttw21[,c("pndwt","pnmodwt")]*length(zz0)
 ttw22=ttw22[,c("pndwt","pnmodwt")]*length(zz0)
 ttw23=ttw23[,c("pndwt","pnmodwt")]*length(zz0)
 ttw24=ttw24[,c("pndwt","pnmodwt")]*length(zz0)
 ttw25=ttw25[,c("pndwt","pnmodwt")]*length(zz0)
 ttw26=ttw26[,c("pndwt","pnmodwt")]*length(zz0)
 ttw27=ttw27[,c("pndwt","pnmodwt")]*length(zz0)
 ttw28=ttw28[,c("pndwt","pnmodwt")]*length(zz0)
 ttw29=ttw29[,c("pndwt","pnmodwt")]*length(zz0)
 ttw30=ttw30[,c("pndwt","pnmodwt")]*length(zz0)
 ttw31=ttw31[,c("pndwt","pnmodwt")]*length(zz0)
 ttw32=ttw32[,c("pndwt","pnmodwt")]*length(zz0)
 ttw33=ttw33[,c("pndwt","pnmodwt")]*length(zz0)
 ttw34=ttw34[,c("pndwt","pnmodwt")]*length(zz0)
 ttw35=ttw35[,c("pndwt","pnmodwt")]*length(zz0)
 ttw36=ttw36[,c("pndwt","pnmodwt")]*length(zz0)
 ttw37=ttw37[,c("pndwt","pnmodwt")]*length(zz0)
 ttw38=ttw38[,c("pndwt","pnmodwt")]*length(zz0)
 ttw39=ttw39[,c("pndwt","pnmodwt")]*length(zz0)
 ttw40=ttw40[,c("pndwt","pnmodwt")]*length(zz0)
 ttw41=ttw41[,c("pndwt",TABLE 9: R code for CUSUM analysis of wavelet coefficients using testing.hov
Appendix B

The R functions `mult.loc` and `testing.hov` from the package waveslim (Whitcher (2013)) are reproduced below.

```r
[1] "mul_list"
function (dwt.list, modwt.list, wf, level, min.coef, debug) 
{
Nj <- length(dwt.list$dwt)
N  <- length(modwt.list$modwt)
crit <- 1.358
change.points <- NULL
if (Nj > min.coef) {
  P <- cumsum(dwt.list$dwt^2)/sum(dwt.list$dwt^2)
  test.stat <- pmax((1:N)/N - (1:Nj)/Nj, (1:Nj)/(Nj - 1) - P)
  loc.dwt <- (1:Nj)[max(test.stat) == test.stat]
  P  <- cumsum(modwt.list$modwt^2)/sum(modwt.list$modwt^2)
  loc.stat <- pmax((1:N)/N - (1:N)/(N - 1), (1:N)/(N - 1) - P)
  loc.modwt <- (1:N)[max(loc.stat) == loc.stat]
if (test.stat > sqrt(2) * crit/sqrt(Nj)) {
  if (debug)
    cat("Accepted!", fill = TRUE)
  if (debug)
    cat("Going left; using", dwt.list$left, " to ",
    loc.dwt + dwt.list$left - 1, ", ...
    temp.dwt.list <- list(dwt = dwt.list$dwt[1:(loc.dwt -
    1)], left = loc.dwt + dwt.list$left - 1)
  temp.modwt.list <- list(modwt = modwt.list$modwt[1:(loc.modwt -
    1)], left = loc.modwt + modwt.list$left - 1)
  change.points <- rbind(c(level, test.stat, loc.dwt +
    dwt.list$left, loc.modwt + modwt.list$left),
    Recall(temp.dwt.list, temp.modwt.list, wf, level,
    min.coef, debug))
  if (debug)
    cat("Going right; using", loc.dwt + dwt.list$left +
    1, ", to", dwt.list$right, ", ...
    temp.dwt.list <- list(dwt = dwt.list$dwt[(loc.dwt +
    1):Nj], left = loc.dwt + dwt.list$left + 1, right = dwt.list$right)
}
```
temp.modwt.list <- list(modwt = modwt.list$modwt[(loc.modwt + 1):N], left = loc.modwt + modwt.list$left + 1, right = modwt.list$right)
change.points <- rbind(change.points, Recall(temp.dwt.list, temp.modwt.list, wf, level, min.coef, debug))
}
else if (debug)
cat("Rejected!", fill = TRUE)
}
else if (debug)
cat("Sample size does not exceed ", min.coef, "!", sep = "", fill = TRUE)
return(change.points)
}<environment: namespace:waveslim>

##testing.hov
function (x, wf, J, min.coef = 128, debug = FALSE)
{
  n <- length(x)
  change.points <- NULL
  x.dwt <- dwt(x, wf, J)
  x.dwt.bw <- brick.wall(x.dwt, wf, method = "dwt")
  x.modwt <- modwt(x, wf, J)
  x.modwt.bw <- brick.wall(x.modwt, wf)
  for (j in 1:J) {
    cat("##### Level ", j, " #####", fill = TRUE)
    Nj <- n/2^j
    dwt.list <- list(dwt = (x.dwt.bw[[j]])[!is.na(x.dwt.bw[[j]])],
                     left = min((1:Nj)[!is.na(x.dwt.bw[[j]])]) + 1, right =
                     sum(!is.na(x.dwt.bw[[j]])
    modwt.list <- list(modwt = (x.modwt.bw[[j]])[!is.na(x.modwt.bw[[j]])],
                        left = min((1:n)[!is.na(x.modwt.bw[[j]])]) + 1, right =
                        sum(!is.na(x.modwt.bw[[j]])
    if (debug)
      cat("Starting recursion; using", dwt.list$left, "to",
           dwt.list$right - 1, "...
    change.points <- rbind(change.points, mult.loc(dwt.list, modwt.list, wf, j, min.coef, debug))
  }
  dimnames(change.points) <- list(NULL, c("level", "crit.value",
                                        "loc.dwt", "loc.modwt"))
  return(change.points)
}<environment: namespace:waveslim>