ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Bachelor Thesis Econometrics and Operations Research

Testing for equality of simultaneously estimated extremal indices

Name student: Job Hölscher
Student ID number: 449506jh

Supervisor: prof. dr. C. Zhou
Second assessor: J.A. Oorschot

Date final version: July 7, 2019

Abstract

Ferro and Segers (2003) proposes a method to estimate the extremal index. In this paper, we propose a test for comparing the values of simultaneously estimated extremal indices for the left and right tails. We use a simulation studies to examine the performance of different estimators for the extremal index. We use a bootstrapping method for the interexceedance times of extreme values to construct confidence intervals of the estimators. For the new test, we use the 'Stationary Bootstrap' (Politis and Romano (1994)) to obtain the standard error of the estimated difference between the two extremal indices. We use the research on the extremal index in Chernick et al. (1991) with our own choice of process in a simulation studies to assess the power of the test. We apply the proposed methods to temperature data and stock returns.

Contents

1 Introduction

Extreme values in financial returns can generate considerable profit or tremendous losses. Such extreme values may occur in clusters. With understanding how extremes are grouped and how these groups behave, investors can predict the occurrence of the next extreme value more accurately. To model clusters of extremes, we will consider the approach in Ferro and Segers (2003). Ferro and Segers (2003) proposes a method for estimating the 'extremal index'. The extremal index indicates the serial dependence between extreme values and has implications for cluster characteristics. Ferro and Segers (2003) suggests estimation methods for the extremal index along with a bootstrap method to assess the confidence interval for the estimates. The goal of this paper is to simultaneously analyze both the clusters of positive and negative extremes and to test on their equality. The similarity of the extremal indices provides insight into the way low and high values tend to cluster. The research question is: is the extremal index for negative extremes equal to the extremal index of positive extremes of a single series?

We will first replicate the method for estimating the extremal index as described in Ferro and Segers (2003) and then extend the research with a method for assessing two extremal indices simultaneously and a testing procedure for comparing these extremal indices. We apply the methods and the extension to both the data in Ferro and Segers (2003) (temperature data in Wooster, Ohio) and returns of the S&P500 index. To test for equality of the estimates with a regular t-test, one way is to assume independence between the variances of the estimates for both extremal indices. However, this assumption might not be valid, since the two estimates are drawn from the same series. Therefore, we apply a 'block bootstrapping' method to assess similarity between the estimates. Politis and Romano (1994) proposes the method of Stationary bootstrapping, which provides us with a way to 'block bootstrap' from dependent data, such as a time series. Using the block bootstrapping, we propose a new test for equality of the estimates. We investigate the power of this test by a simulation studies.

Zhang et al. (2011) find an increasing amount of extreme temperature periods over the past decades. More frequent fluctuations in temperatures cause extreme temperature periods to be shorter. Furthermore, greenhouse gasses are known to drive up temperatures over time. Since we use a constant threshold value for the entire sample period, we expect cold periods to occur more often in the first half of the sample period and warm periods to occur more frequently in the second half. We assume longer and less frequent extreme value clusters in the early years, thus we expect cold periods to last longer than warm periods. We find that extreme cold periods indeed are clustered more heavily than warm periods during the winter months of the 20th century.

Oswiecimka et al. (2008) analyzes the fractal properties of positive and negative returns. They find, for the German Dax30 index, bear markets to be more persistent than bull markets. In other words, periods with negative returns tend to last longer than periods of positive returns. This indicates that negative extremes may have larger clusters than positive extremes. Furthermore, Wu (2015) studies the asymmetry in volatility in equity markets. Wu (2015) finds that volatility feedback is both economically and statistically significant. Large negative returns have a larger impact on volatility than positive returns do. Higher volatility means observations exceed the positive or negative threshold more frequently, implying that negative extremes tend to be clustered more than positive extremes. We find, however, no significant results for the difference in clustering of extreme positive and negative daily log returns of the S&P 500 index.

We state the methods used for estimating the extremal index, a simulation procedure, the two bootstrapping methods, a way of testing for similarity of extremal indices in one series and a way to assess the power of this test in section 2. We describe the data for the applications in section 3 and present our findings of the simulation, the power of our test and applications in section 4. Section 5 concludes the applications and the performance of our test. This paper contributes to current literature by proposing a test for equality of two simultaneously estimated extremal indices, along with the power of this test, and by applying the methods to a financial time series.

2 Methodology

To estimate the extremal index, Ferro and Segers (2003) defines a strictly stationary sequence of random variables by $\{\xi_n\}_{n\geq 1}$, for which the marginal distribution function is F. Define $M_{k,l} = \max\{\xi_i : i = k+1, ..., l\}$ and $M_n = M_{0,n}$, where $0 \leq k < l$ and $n \geq 1$. If there exists a sequence $\{u_n\}_{n\geq 1}$, such that, for any $\tau > 0$, as $n \to \infty$,

$$n\{1 - F(u_n)\} \to \tau \quad \text{and} \quad P(M_n \leq u_n) \to \exp(-\theta\tau), \tag{1}$$

then the parameter $\theta \in [0, 1]$ is called the 'extremal index' of the sequence. If $\theta = 1$, exceedances of a certain large threshold are independent (there is no clustering of extreme values in the limit). However, if $\theta < 1$, extreme values (threshold exceedances) do tend to cluster in the limit. Hence, an increase in θ points towards a decrease in dependence between extreme values in the considered sequence. Ferro and Segers (2003) defines the interexceedance times of the process exceeding a threshold u, by:

$$T(u) = \min\{n \geq 1 : \xi_{n+1} \geq u\}. \tag{2}$$

This interexceedance time is in distribution given by

$$P\{T(u) > n\} = P(M_{1,n+1} \leq u \mid \xi_1 > u). \tag{3}$$

Theorem 1 in Ferro and Segers (2003) states that, as $n \to \infty$,

$$P\{(1 - F(u_n)T(u_n) > t\} \to \theta exp(-\theta t) \qquad for \quad t > 0. \tag{4}$$

In other words,

$$(1 - F(u))T(u) \xrightarrow{d} T_\theta \qquad as \quad u \uparrow \omega \quad and \tag{5}$$

$$T_\theta \sim (1 - \theta)\epsilon_0 + \theta\mu_\theta, \tag{6}$$

where $\omega = sup\{x : F(x) < 1\}$, ϵ_0 is the degenerate distribution at 0 and $\mu_\theta \sim exp(\theta)$. We remark that the distribution of T_θ is independent of the choice of threshold u.

In order to estimate the extremal index, Ferro and Segers (2003) defines $N = N_n(u) = \sum_{i=1}^{n} I(\xi_i > u)$ to be the number of observations above the threshold u and S_i to be the $i-$th exceedance time. It follows that the *observed* interexceedance times are $T_i = S_{i+1} - S_i$ for $i = 1, ..., N - 1$.

Ferro and Segers (2003) proposes the following 'intervals estimator' for θ:

$$\theta_n(u) = \begin{cases} 1 \wedge \hat{\theta}_n(u), & \text{if} \quad \max\{T_i : 1 \le i \le N - 1\} \le 2, \\ 1 \wedge \hat{\theta}^*_n(u), & \text{if} \quad \max\{T_i : 1 \le i \le N - 1\} > 2, \end{cases}$$

where $\hat{\theta}_n(u)$ and $\hat{\theta}^*_n(u)$ are defined as follows:

$$\hat{\theta}_n(u) = \frac{2(\sum_{i=1}^{N-1} T_i)^2}{(N-1)\sum_{i=1}^{N-1} T_i^2}, \tag{7}$$

$$\hat{\theta}^*_n(u) = \frac{2(\sum_{i=1}^{N-1} (T_i - 1))^2}{(N-1)\sum_{i=1}^{N-1} (T_i - 1)(T_i - 2)}. \tag{8}$$

Estimator $\hat{\theta}^*_n(u)$ makes sure that the smallest interexceedance times have no impact. *Theorem 2* in Ferro and Segers (2003) states that, under some mild conditions, $\theta_n(u_n) \xrightarrow{p} \theta$.

Another way of estimating the extremal index is through runs declustering, which assigns exceedances to clusters based on their interexceedance times. Run length r defines the maximum amount of days in between threshold exceedances for which the exceedances belong to the same cluster. Ferro and Segers (2003) proposes a runs estimator for the extremal index:

$$\hat{\hat{\theta}}_n(u) = N^{-1}\bigg(1 + \sum_{i=1}^{N-1} I(T_i > r)\bigg). \tag{9}$$

We vary the threshold, which results in a different number of exceedances, N. We disregard thresholds for which we observe fewer than 10 exceedances. Also, we do not use thresholds for which more than approximately 10% of the data exceeds its value, because the asymptotic properties do not hold when the observations do not belong to the 'tail'.

2.1 Bootstrap from interexceedance times

In order to assess the confidence interval for the estimate of θ, Ferro and Segers (2003) uses a bootstrapping method. The method is applied to the observed interexceedance times in the following manner: we order all $N-1$ interexceedance times from small to big: $T_{(1)}, ..., T_{(N-1)}$. We assume the $C-1 = \lfloor \theta N \rfloor$ largest interexceedance times to be intercluster times. If we use the intervals estimator for θ in determining C, Ferro and Segers (2003) states that asymptotic theory justifies an entirely automatic declustering procedure. The $N-1-C$ other interexceedance times shall be considered intracluster times. When bootstrapping, these intracluster times will be grouped, such that extreme value clusters are not separated accidentally. For each group, the intracluster times have appointed to it a set of exceedance times. First, we re-sample the intercluster times with repetition until $C-1$ intercluster times, along with their exceedance times, are 'chosen'. Then, we bootstrap from the groups of intracluster times and intercalate these between the intercluster times, also with the possibility to repeat, until C clusters are 'chosen'. Since cluster sizes differ, for the bootstrapped series, the total amount of threshold exceedances might not be exactly equal to N. This bootstrap replication of the process yields the possibility to re-estimate θ. If we repeat this multiple times (further denoted as B times), we can determine a confidence interval for the estimate of θ from the B re-estimates. We construct a symmetric confidence interval using the empirical mean and variance of the B re-estimates under the assumption of normality.

2.2 Simulation for estimation

In order to test the methods for varying thresholds, we simulate data on which we test the performance of the intervals estimator, the runs estimator and bootstrapping from the interexceedance times according to the method in Ferro and Segers (2003). We define the data generating process (DGP) to be the following max-autoregressive process, $\{\xi_n\}_{n\geq 1}$:

$$\xi_1 = W_1/\theta, \tag{10}$$

$$\xi_n = \max\{(1-\theta)\xi_{n-1}, W_n\} \quad for \quad n \geq 2, \tag{11}$$

where $\theta \in (0,1]$ and W_n, for $n \geq 1$, are independent unit Fréchet random variables. The *true* extremal index is θ in $\{\xi_n\}_{n\geq 1}$. We simulate 100 sequences of length 5000 for θ equal to 0.25, 0.5 and 0.75. The thresholds are chosen such that there are N exceedances, starting with 10 exceedances and a step length of 10. For

each threshold, the extremal index is estimated by the intervals estimator and the runs estimator with run lengths 1,5 and 9. The performance is measured by the Root Mean Squared Error (RMSE) and the coverage probability using a 95% confidence interval, for which we use the bootstrapping procedure as described in section 2.1 with $B = 50$.

2.3 New test for equality of estimators

We will compare two estimates for extremal indices in one time series: the extremal index for extreme negative values (θ_{neg}) and the extremal index for extreme positive values (θ_{pos}). We are interested in testing for equality of these estimates;

$$H_0 : \quad \theta_{pos} = \theta_{neg}. \tag{12}$$

We define $\hat{\theta}_{neg}$ to be the estimated extremal index using all negative returns above a certain threshold and $\hat{\theta}_{pos}$ the estimated extremal index using all positive returns above a certain threshold, such that the amount of exceedances is identical in the estimation procedure.

For testing, we need the asymptotic distribution of $\hat{\theta}_{pos} - \hat{\theta}_{neg}$. A naive way is to assume independence between the two estimators. However, this assumption is questionable. An observation cannot belong to the set of negative extremes and to the set of positive extremes simultaneously, for example. In order to avoid this assumption, we use block bootstrapping to construct the 95% confidence interval for $\hat{\theta}_{pos} - \hat{\theta}_{neg}$, while maintaining possible serial correlation. Politis and Romano (1994) proposes the 'stationary bootstrap' method for resampling from serially dependent data, which we will describe in the next section. For each time we resample, we calculate the statistic $\hat{\theta}_{pos} - \hat{\theta}_{neg}$. If we repeat this B times, we can check whether zero is embedded in the confidence interval of the statistic for different thresholds. The confidence interval is symmetric, based on the empirical mean and variance of the B re-estimates of $\hat{\theta}_{pos} - \hat{\theta}_{neg}$, where we use $B = 500$. We calculate the p-values for the test using different thresholds and different N. Since these tests are dependent, we cannot use the regular Bonferroni method (Bland and Altman (1995)). To cope with this dependence, Sarkar and Chang (1997) proposes the 'Simes method'. Sarkar and Chang (1997) shows that for many positively dependent multiple testing situations, the Simes method controls for the probability of Type I error (incorrect rejection of true hypothesis).

2.4 Stationary bootstrap

Under the mild assumptions of weak dependence and stationarity for the data, we apply the stationary bootstrap method. Politis and Romano (1994) proposes the stationary bootstrap to construct a resampled series by bootstrapping from the original data as follows: we select a random observation from the original data $X_1, .., X_n$ to start with the pseudo time series X_1^*. After obtaining X_{i-1}^* for $2 \leq i \leq n$, with probability

p, we pick X_i^* at random from the original data. Given that X_{i-1}^*=X_j, with probability $1-p$, we choose X_i^* =X_{j+1}, so that we pick the 'next' observation from the original data. Note that we pick values with repetition. This procedure generates a bootstrapped sample with sample size n.

By construction, we take blocks of consecutive observations. The length of each block, L_t, is independent and identically distributed (i.i.d.) following the geometric distribution with a parameter p;

$$P(L_t = m) = (1-p)^{m-1}p \qquad for \quad t = 1, ..., M, \tag{13}$$

where M is the amount of blocks, $\sum_{t=1}^{M} L_t = n$. In section 3, we check whether the Stationary bootstrap method preserves possible autocorrelation in the original data.

2.5 Power of the new test

Before we apply the test to real data, we investigate its power with a simulation studies. We generate data which contains dependence in both left and right tails. Chernick et al. (1991) calculates the extremal index for a moving average representation as follows: denote

$$F_j = \sum_{i=-\infty}^{\infty} c_i Z_{j-i}, \quad 1 \leq j \leq n, \tag{14}$$

where $\{Z_i\}_{i=-\infty}^{+\infty}$ are independent and identically distributed with a tail index α. Define $p \in [0,1]$ as:

$$p = \lim_{x\to\infty} \frac{P[Z_1 > x]}{P[|Z_1| > x]}. \tag{15}$$

Under some mild conditions on c_i and Z_{j-i}, the extremal index of the positive extreme values is given by:

$$\theta_{pos} = \frac{p[c^+]^\alpha + q[c^-]^\alpha}{p\sum_{i=-\infty}^{\infty}[c_i^+]^\alpha + q\sum_{i=-\infty}^{\infty}[c_i^-]^\alpha}, \tag{16}$$

where $q = 1-p$, $c_i^+ = c_i \vee 0$, $c_i^- = -c_i \vee 0$, $c^+ = \bigvee_i c_i^+$ and $c^- = \bigvee_i c_i^-$. We can calculate the extremal index of the negative extreme values by using the negated values of $F_1, .., F_n$. Note that using the negated values of $F_1, .., F_n$ boils down to expression (16), with swapping the parameters p and q. We consider an ARMA(1,1) model:

$$X_t - \rho X_{t-1} = Z_t + \eta Z_{t-1} \quad \text{for} \quad t \geq 1, \tag{17}$$

which has the moving average representation:

$$X_t = \sum_{i=0}^{\infty} c_i Z_{j-i}, \quad j \geq 1, \tag{18}$$

where $c_0 = 1$ and $c_i = \rho^{i-1}(\rho + \eta)$ for $i \geq 1$. Here, ρ is the autoregressive parameter and η is the moving average parameter. For stationarity, we impose the restrictions $0 < \rho < 1$, and $-1 < \eta < -\rho$. Using these

values for c_i in (16) gives the following extremal indices for positive and negative extreme values:

$$\theta_{pos} = \frac{p + q|\rho + \eta|^\alpha}{p + q|\rho + \eta|^\alpha (1-\rho)^{-\alpha}} \quad \text{and} \quad \theta_{neg} = \frac{q + p|\rho + \eta|^\alpha}{q + p|\rho + \eta|^\alpha (1-\rho)^{-\alpha}}. \tag{19}$$

Remark that the difference between θ_{pos} and θ_{neg} is determined by the values of p and q.

In order to generate data with unbalanced tails, we define

$$Z_j = \begin{cases} Y_j & \text{if} \quad Z_j \geq 0, \\ cY_j & \text{if} \quad Z_j < 0, \end{cases}$$

where Y_j follows the Student's t distribution with degrees of freedom α and c is a positive constant. If $c > 1$, the distribution is negatively skewed. *Proof 1* in the Appendix shows that $p = (1 + c^\alpha)^{-1}$. Note that $c = 1$ corresponds to $p = q = 0.5$, hence $\theta_{pos} = \theta_{neg}$. As we increase c, we increase the difference between θ_{pos} and θ_{neg}. From (19), we derive that as $p \to 0^+$, $(\theta_{pos} - \theta_{neg}) \to -\rho^2$. In order to generate a considerable difference between the 'true' values of the extremal indices, we choose a reasonably large value for the autoregressive parameter ($\rho = 0.6$), η is fixed at -0.9 and we use $\alpha = 2$. For each value of c ($c = 1, .., 10$), we calculate the fraction of times where zero is embedded in the 95% confidence interval of $\hat{\theta}_{pos} - \hat{\theta}_{neg}$. In other words, we calculate the fraction of times where the null hypothesis is rejected. For $c = 1$, this is the size and for $c \neq 1$ the power of the test. The confidence interval is constructed using the block bootstrapping procedure with B re-estimates of $\hat{\theta}_{pos} - \hat{\theta}_{neg}$, where $B = 50$. For the simulation, we use 100 ARMA(1,1) processes with length 2000.

3 Data for applications

3.1 Wooster temperatures

The data used in Ferro and Segers (2003) consists of negated daily minimum temperatures, recorded to the nearest degree Fahrenheit, at Wooster, Ohio. We choose this weather station because of the long-term daily data that is available. We retrieve the data from https://www.ncdc.noaa.gov/. For this research, we use daily data from January 1st, 1900 until December 31st, 1999. Coles et al. (1994) find that the series is approximately stationary over the winter months. Therefore we use the months December, January and February in our analysis. Table 1 shows the first five partial autocorrelations of the Temperature data. We observe a similar pattern for the resampled temperatures and the original data. Therefore, the stationary bootstrap method preserves serial correlation in the temperature data.

3.2 S&P500 Log returns

For the second application, we will use the daily log returns of the S&P500 index fund, retrieved from Bloomberg. We use the S&P500 index from January 4th, 1999 until December 31st, 2018. We observe a comparable pattern for the partial autocorrelations of the squared log returns in Table 1, which indicates that the Stationary bootstrap method also preserves the autocorrelation from the original squared log returns of the S&P 500.

Table 1: Partial autocorrelations of original and resampled data

	Wooster temperatures	Resampled temperatures
1^{st}	-0.105	-0.122
2^{nd}	-0.221	-0.206
3^{rd}	-0.151	-0.156
4^{th}	-0.110	-0.115
5^{th}	-0.118	-0.116

	Log S&P 500 returns	Resampled returns
1^{st}	-0.606	-0.566
2^{nd}	-0.240	-0.314
3^{rd}	-0.251	-0.213
4^{th}	-0.284	-0.218
5^{th}	-0.172	-0.143

The first five partial autocorrelations for the original data and the average values of the first 10 series constructed by 'stationary bootstrapping' from the original data for both applications; the data is squared for the log returns series

4 Results

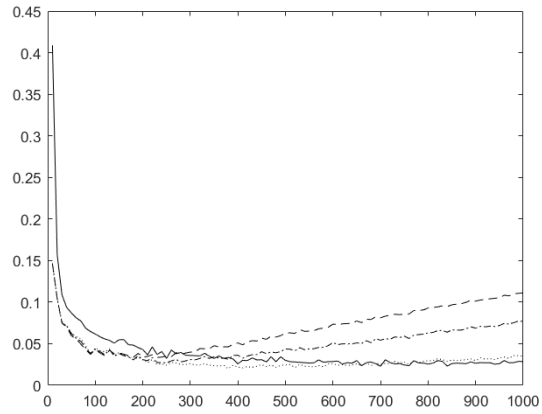
4.1 Simulation results for estimation

Figure 1 shows the simulation results of the intervals estimator and the runs estimator (for run lengths 1,5 and 9). The left three figures in Figure 1 show that for each level of serial dependence in the series, all runs estimators have a lower RMSE than the intervals estimator for very high thresholds (few exceedances). On the other hand, the intervals estimator outperforms the runs estimators for (slightly) lower thresholds, especially with little dependence present in the DGP. The latter is also supported by the right figures, where the intervals estimator is robust for decreasing thresholds. The runs estimators, unlike the intervals estimator, are static in the way that they predefine a run length. With a decreasing threshold, the 'real' run length can

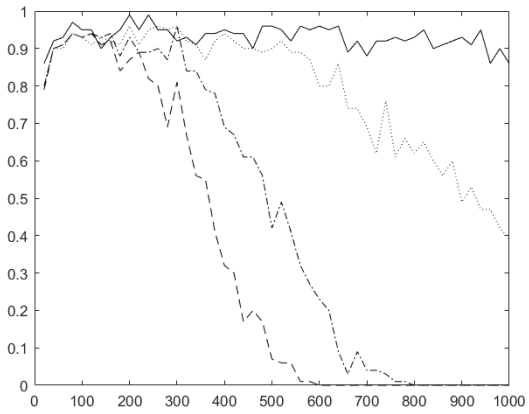
vary, making the runs estimators inaccurate. The true value of θ is contained in approximately 95% of the confidence intervals for the intervals estimator. Therefore, the bootstrapping method is an accurate method for determining the confidence interval for the intervals estimator.

4.2 Simulation results for testing

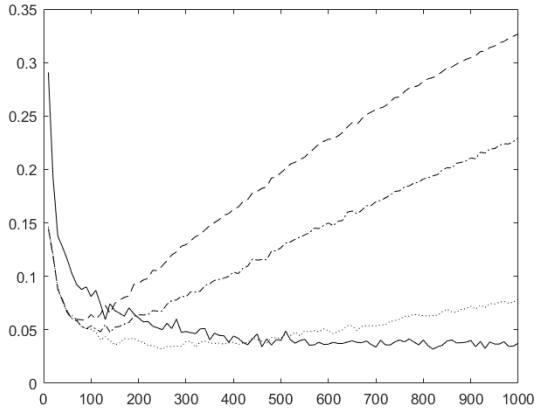
Figure 2 shows the results of the size and power of our new test for equality of two simultaneously estimated extremal indices. We observe the power to increase as c increases, especially in the Figures 2(b), 2(c) and 2(d). This is due to the fact that the 'true' difference between θ_{pos} and θ_{neg} increases with c. Figure 2 shows a 0% rejection rate when $c = 1$, except for the intervals estimator for low thresholds. Our test does not reject too often when the null hypothesis is true, but the size is only appropriate for the intervals estimator with 300 exceedances. Figure 2(a) shows that the test does not have much power when we use 50 exceedances for estimation. Furthermore, the power of our test is consistently lower when using the runs estimator. This is potentially due to the low coverage probability of the runs estimator for low thresholds. Due to the comparable coverage probability and root mean squared error for high thresholds, the runs estimator and intervals estimator have comparable power in 2(a) and 2(b). Overall, Figure 2(d) shows a desirable pattern for the intervals estimator with 300 exceedances.



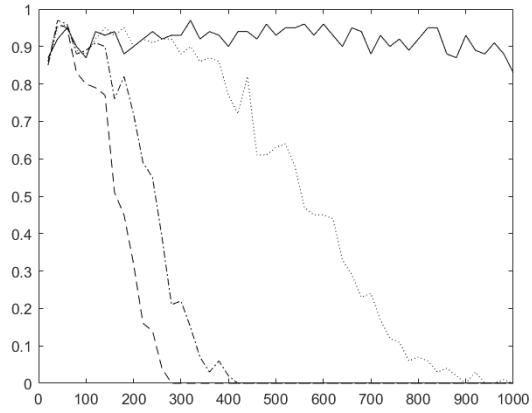
(a)



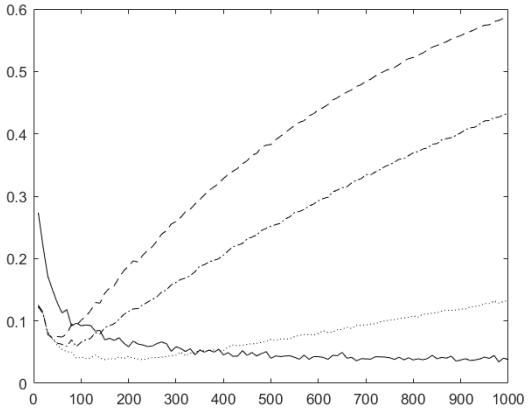
(b)



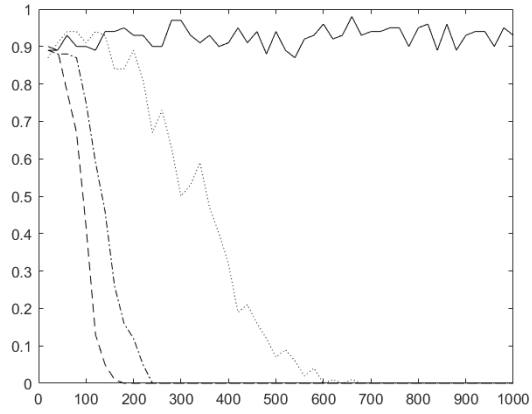
(c)



(d)



(e)



(f)

Figure 1: Performance summaries of the intervals estimator (—) and runs estimator, with run lengths $1(\cdots), 5(\cdot - \cdot)$ and 9(- - -). In the left three figures, the Root Mean Squared Error (RMSE) is depicted and in the right figures, the coverage probability for different amounts of exceedances is summarized. The coverage probability is the empirical fraction of 100 confidence intervals in which the true theta is embedded

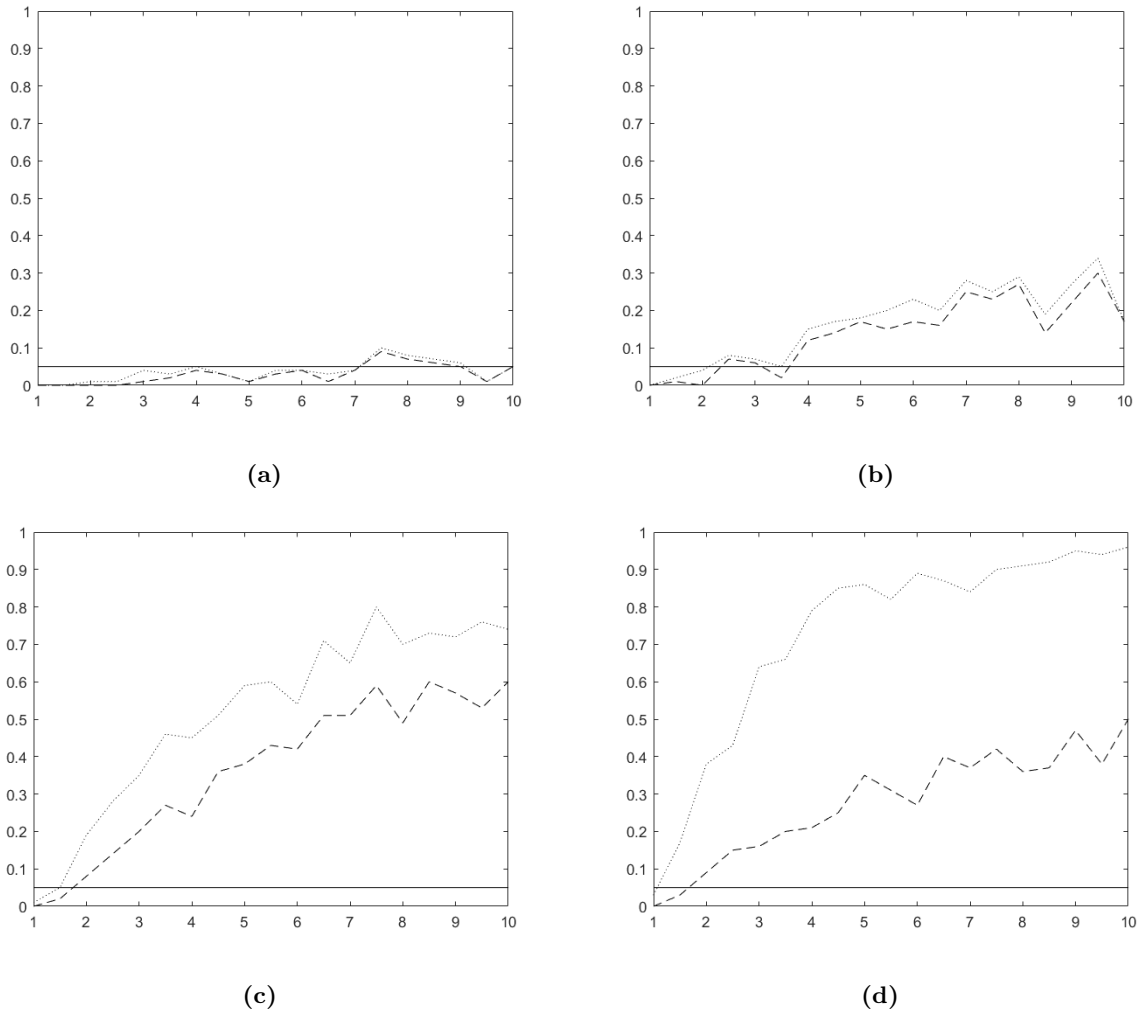


Figure 2: Fraction of times that $H_0 : \theta_{pos} - \theta_{neg} = 0$ is rejected for (a) 50, (b) 100, (c) 200 and (d) 300 exceedances for different values of c. The values of ρ, η and α are fixed at $0.6, -0.9$ and 2, respectively. Each simulated series has length $n = 2000$. The runs estimator with run length 5 (- - -) and the intervals estimator ($\cdots$) are used and the horizontal line in each graph is the 5% rejection rate

4.3 Temperature data results

Figure 3 shows estimates of cluster characteristics with different thresholds u (u=0.5,1.5,...,14.5) for the Temperature data. Figure 3(a) depicts the intervals estimator for the extremal index of negated temperatures, which we observed in the right tail simulation results to be accurate for low thresholds (5 to 8). It therefore points towards an extremal index of around 0.5. Figure 3(b) summarizes the runs estimator for the extremal index, with run length 5, for the same thresholds. The runs estimator performs best for high thresholds. Hence, it indicates an extremal index of 0.6. These values are similar to the findings in Ferro and Segers (2003). Figure 3(c) shows the mean cluster excess for different thresholds. We define the mean cluster excess,

ζ_u, as:

$$\zeta_u = C^{-1} \sum_{i=1}^{n} \max(0, \xi_i - u) \quad for \quad u = 0.5, ..., 14.5, \tag{20}$$

where n is the amount of observations. For constructing the confidence interval, the variance of ζ_u is estimated with bootstrapping from the interexceedance times, where we record the clusters of exceedance times with the clusters of exceeding values. The mean cluster excess in 3(c) is decreasing as the threshold increases. The graph flattens out around 8°F. Finally, recall that each bootstrap process has a run length, r. For each threshold value, the variance in r indicates the variation ignored by using arbitrary run lengths. Figure 3(d) depicts the estimated run length. The run length of each (bootstrapped) process is equal to the biggest intracluster time, $T_{(C)}$. Especially for low thresholds, a run length of around 7 is suggested with the interpretation that two very cold days that are no more than one week apart belong to the same cold period.

Figure 4 shows the confidence intervals of $\hat{\theta}_{pos} - \hat{\theta}_{neg}$ for different thresholds. Using the asymptotic mean and variance of the re-estimates, the Simes method yields a p-value of 0.000 for both the runs estimator and the intervals estimator. Hence at least one null hypothesis is rejected. It shows that, for any threshold for which there are more than 150 exceedances, zero is not embedded in the confidence intervals. The results of the power of our test indicate that the test with neither estimation method rejects too often. Hence, we reject equality of the extremal indices. The threshold that corresponds to 150 exceedances is 43°F for positive extremes and -8°F for negative extremes. We find that the negative extreme values are clustered more heavily than the positive extreme values. In other words, cold winter periods are clustered more than warm winter periods. This is in line with our hypothesis.

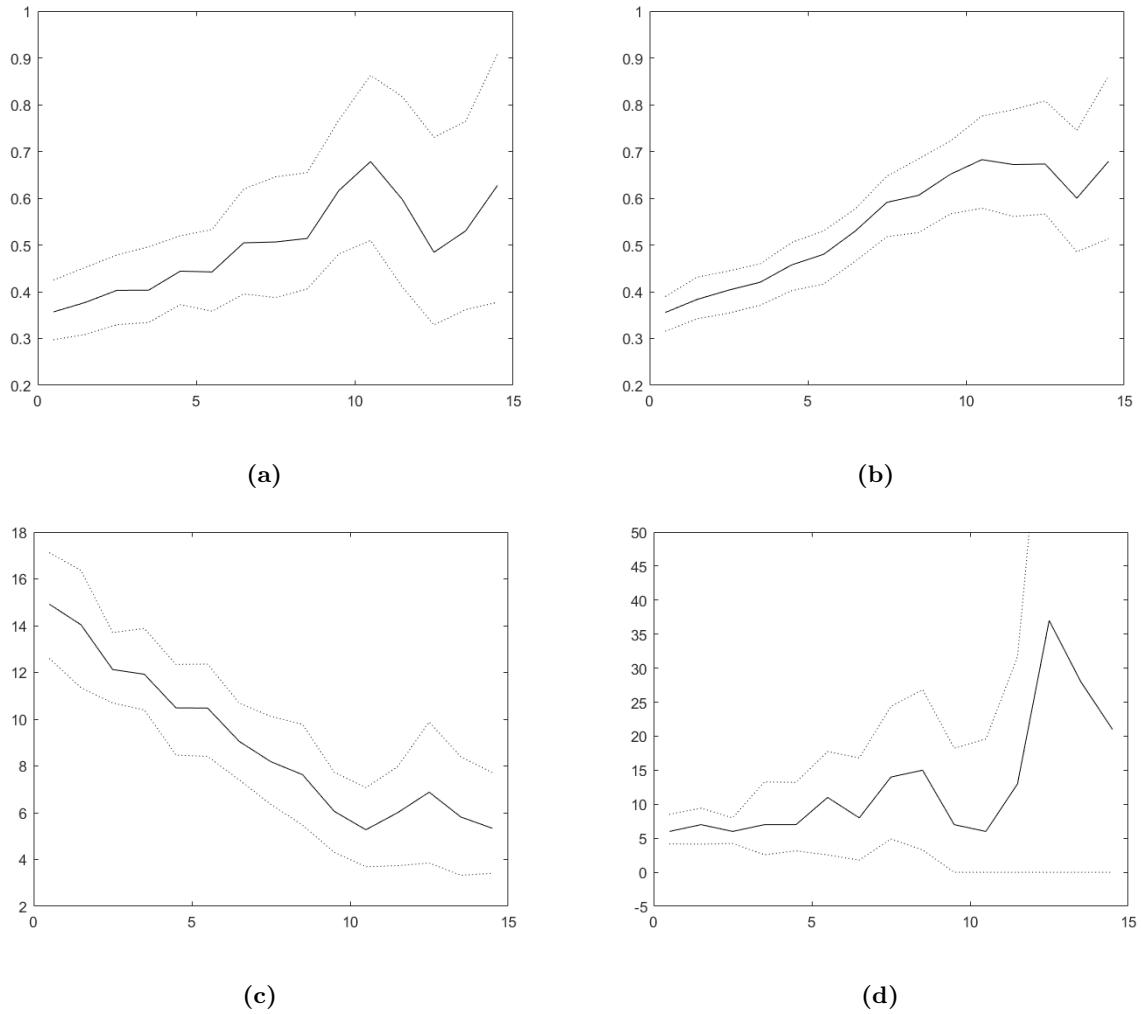
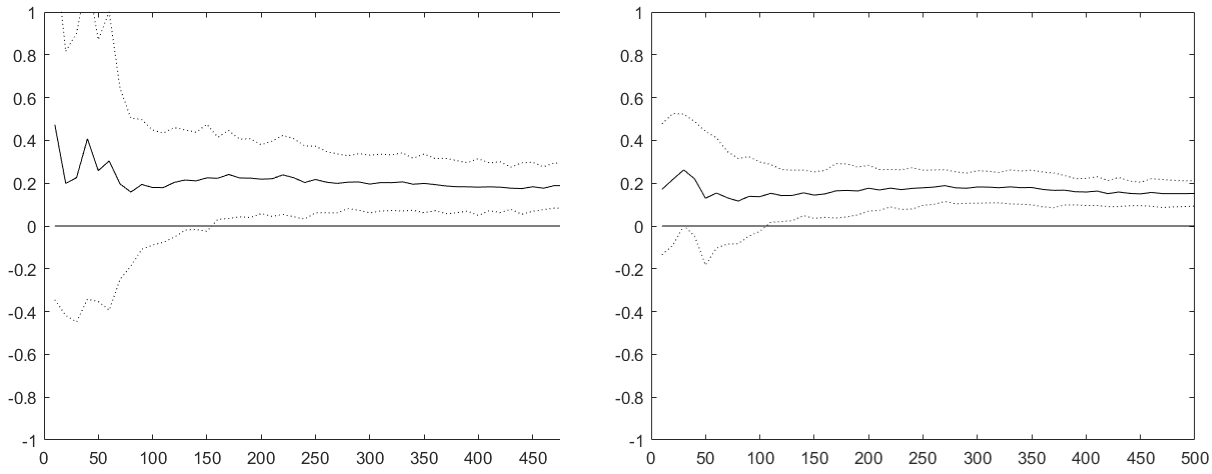


Figure 3: Estimates of cluster characteristics of the Wooster temperatures with threshold on the horizontal axis in negated degrees °F: (a) intervals estimator, (b) runs estimator (run length 5), (c) mean cluster excess and (d): run length; $(\cdots)$ is the 95% confidence interval constructed by bootstrapping from interexceedance times



(a) **(b)**

Figure 4: Estimates of the test statistic $\hat{\theta}_{pos} - \hat{\theta}_{neg}$ (—) along with 95% intervals ($\cdots$), constructed by block bootstrapping from the temperature data, where (a) is estimated with the intervals estimator and (b) is estimated using the runs estimator with run length 5. The value under the null hypothesis (12) is the horizontal line at zero

4.4 Stock returns results

In this section, we apply the extremal index estimators in Ferro and Segers (2003) and the simultaneous estimation on the log returns of the stock index S&P 500. Figure 5 shows the estimates of the extremal index for positive extreme values using the intervals estimator in 5(a) and the runs estimator with run length 5 in 5(b). It further shows estimates of $\hat{\theta}_{pos} - \hat{\theta}_{neg}$ for the intervals estimator in 5(c) and the runs estimator with run length 5 in 5(d). Following simulation for estimation, the intervals estimator performs well for low thresholds and the runs estimator performs best for high thresholds. Hence, the intervals estimator points towards a value of 0.4 for the positive extreme values and the runs estimator towards a value of 0.6. The Simes method yields a p-value of around 0.47 for the intervals estimator and a p-value of 0.002 for the runs estimator. Figure 5(d) shows that zero is not embedded in the confidence interval for low thresholds using the runs estimator. However, simulation for estimation results indicate a very low coverage probability of the runs estimator with run length 5 for low thresholds. We therefore disregard the runs estimator's p-value. The power of the intervals estimator for low thresholds however has proven to be promising, hence we do not reject the hypothesis of equality for any amount of exceedances, up to 500. Hence, in contrast with our hypothesis, we do not observe significantly heavier clustering of negative extreme stock returns than positive extreme stock returns.

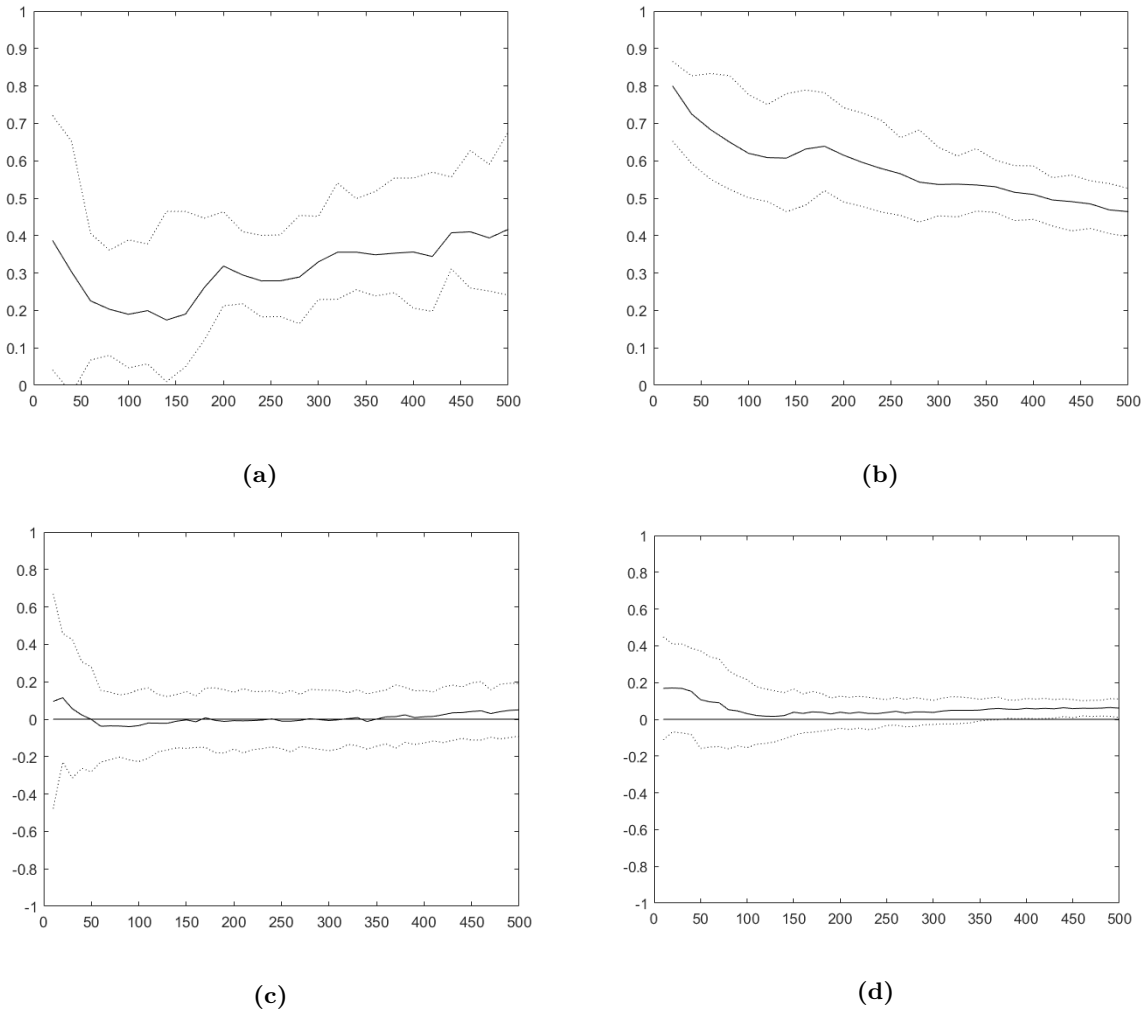


Figure 5: Estimates of (the difference between) extremal indices of the stock data. In the left figures, the intervals estimator is used. The right figures are constructed using the runs estimator with run length 5. The top figures are the estimation of the extremal index for positive extreme values and the bottom figures are the estimations of the test statistic $\hat{\theta}_{pos} - \hat{\theta}_{neg}$. The horizontal axis is the amount of exceedances; $(\cdots)$ is the 95% confidence interval, constructed by either bootstrapping from interexceedance times (top figures) or the Stationary block bootstrap (bottom figures). The value under the null hypothesis (12) is the horizontal line at zero

5 Conclusion

In this study, we investigate the performance of the intervals estimator and runs estimator from Ferro and Segers (2003) using a simulation studies. We propose a new testing procedure for testing for equality of two simultaneously estimated extremal indices using either estimator. We find the performance of this test to be best using the intervals estimator in combination with a low threshold. Furthermore, we apply our methods

to extreme temperatures in Wooster, Ohio, and to the daily log returns of the S%P500 stock. We observe significantly more clustering of cold periods than warm periods during the winter months of the 20th century. Concerning the clustering of extreme daily log returns of the S&P 500 index, we do not find a significant difference between the clustering of extreme positive and negative returns. This indicates that investors do not react differently to extreme negative and extreme positive returns. Extreme negative returns might have more impact on volatility than extreme positive returns, but during the period of high volatility, both high positive and high negative returns occur. Therefore, the average sizes of the clusters of both positive and negative extremes are similar.

In order to extend on this research, the theoretical value of the dependence between θ_{pos} and θ_{neg} could be used to derive a more precise testing procedure. Also, we leave investigation of the robustness of the test to variation in the parameters ρ, η and α to further research.

6 Appendix

6.1 Proof 1

For the right tail, we have $P(Z_1 > t) = P(Y_1 > t)$, and for the left tail, we have $P(Z_1 < -t) = P(Y_1 < -t/c)$. Since $Y_1 \sim t(\alpha)$, the following asymptotic property holds:

$$P(Y_1 < -t/c) \sim P(Y_1 < -t)(1/c)^{-\alpha} \quad \text{as} \quad t \to \infty, \tag{21}$$

where α is the degrees of freedom of the Student's t distribution. By definition,

$$p = \lim_{t\to\infty} \frac{P(Z_1 > t)}{P(|Z_1| > t)} \tag{22}$$

$$= \lim_{t\to\infty} \frac{P(Z_1 > t)}{P(Z_1 > t) + P(Z_1 < -t)} \tag{23}$$

$$= \lim_{t\to\infty} \frac{P(Y_1 > t)}{P(Y_1 > t) + P(Y_1 < -t/c)}. \tag{24}$$

Expressions (21) and (24) imply:

$$p = \lim_{t\to\infty} \frac{P(Y_1 > t)}{P(Y_1 > t) + P(Y_1 < -t)(1/c)^{-\alpha}} \tag{25}$$

$$= \lim_{t\to\infty} \frac{P(Y_1 > t)}{P(Y_1 > t) + P(Y_1 > t)(1/c)^{-\alpha}} \tag{26}$$

$$= (1 + c^{\alpha})^{-1}. \tag{27}$$

References

Bland, J. M. and Altman, D. G. (1995). Multiple significance tests: the bonferroni method. *British Medical Journal*, 310(6973):170.

Chernick, M. R., Hsing, T., and McCormick, W. P. (1991). Calculating the extremal index for a class of stationary sequences. *Advances in applied probability*, 23(4):835–850.

Coles, S. G., Tawn, J. A., and Smith, R. L. (1994). A seasonal markov model for extremely low temperatures. *Environmetrics*, 5(3):221–239.

Ferro, C. A. and Segers, J. (2003). Inference for clusters of extreme values. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(2):545–556.

Oswiecimka, P., Kwapien, J., Drozdz, S., Górski, A., and Rak, R. (2008). Different fractal properties of positive and negative returns. *arXiv preprint arXiv:0803.1374*.

Politis, D. N. and Romano, J. P. (1994). The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313.

Sarkar, S. K. and Chang, C.-K. (1997). The simes method for multiple hypothesis testing with positively dependent test statistics. *Journal of the American Statistical Association*, 92(440):1601–1608.

Wu, G. (2015). The Determinants of Asymmetric Volatility. *Review of Financial Studies*, 14(3):837–859.

Zhang, X., Alexander, L., Hegerl, G. C., Jones, P., Tank, A. K., Peterson, T. C., Trewin, B., and Zwiers, F. W. (2011). Indices for monitoring changes in extremes based on daily temperature and precipitation data. *Wiley Interdisciplinary Reviews: Climate Change*, 2(6):851–870.

MATLAB codes used

```
function [T] = figure1ace(theta)
% FIGURE1ACE plots the RMSE for N ranging from 10 to 1000 (figures 1a, 1c
% and 1e)
%
% theta is the 'true' extremal index in the simulation

RMSE_Int = zeros(100,1);    %initialize values
RMSE_R1 = zeros(100,1);
RMSE_R2 = zeros(100,1);
RMSE_R3 = zeros(100,1);
for Q = 1:100
    N = Q*10;
    intervalsEstimator = zeros(100,1);
    runsEstimator1 = zeros(100,1);
    runsEstimator5 = zeros(100,1);
    runsEstimator9 = zeros(100,1);
    Xisequences = zeros(5000,1000);
    for p = 1:100               %calculate the estimates 100 times
        XiSequences(:,p) = getXi(theta);    %simulate data
        thresholdValue = getThreshold(N, XiSequences(:,p));
        S = zeros(N,1);
        exceedancetime = 1;
        for q = 1:5000
            if XiSequences(q,p) > thresholdValue
                S(exceedancetime) = q;
                exceedancetime = exceedancetime+1;
            end
        end
        T = zeros(N-1,1);
        for m = 1:N-1
            T(m) = S(m+1) - S(m);   %interexceedance times of simulated data
        end

        intervalsEstimator(p) = (getIntervalsEst(T)-theta)^2;   %calculate
            squared error
```

```
35          runsEstimator1(p) = (getRunsestimator(1, N, T)-theta)^2;
36          runsEstimator5(p) = (getRunsestimator(5, N, T)-theta)^2;
37          runsEstimator9(p) = (getRunsestimator(9, N, T)-theta)^2;
38
39      end
40      RMSE_Int(Q) = sqrt(mean(intervalsEstimator)); %calculate the square root
            of the mean of the squared error
41      RMSE_R1(Q) = sqrt(mean(runsEstimator1));
42      RMSE_R5(Q) = sqrt(mean(runsEstimator5));
43      RMSE_R9(Q) = sqrt(mean(runsEstimator9));
44  end
45  plot(RMSE_Int, 'k-') %plot the graphs
46  hold on
47  plot(RMSE_R1, 'k:')
48  hold on
49  plot(RMSE_R5, 'k-.')
50  hold on
51  plot(RMSE_R9, 'k--')
52  end
```

```
1   function [T] = figure1bdf(theta)
2   %FIGURE1BDF generates the coverage probability figures
3   %
4   % theta is the 'true' theta in the simulation
5
6   Coverage_int = zeros(50,1);          %initialize vectors
7   Coverage_runs1 = zeros(50,1);
8   Coverage_runs5 = zeros(50,1);
9   Coverage_runs9 = zeros(50,1);
10  xaxis = zeros(50,1);
11
12  for Q=1:50
13      N = Q*20;
14      xaxis(Q) = Q*20;
15      amountyes_int = 0;
16      amountyes_runs1 = 0;
```

```
17      amountyes_runs5 = 0;
18      amountyes_runs9 = 0;
19
20      for i=1:100                 % generate 100 processes
21          XiSequence = getXi(theta);
22          thresholdValue = getThreshold(N, XiSequence);
23          S = zeros(N,1);
24          exceedancetime = 1;
25
26          for q = 1:5000          % get the extreme values
27              if XiSequence(q) > thresholdValue
28                  S(exceedancetime) = q;
29                  exceedancetime = exceedancetime+1;
30              end
31          end
32
33          T = zeros(N-1,1);
34
35          for m = 1:N-1
36              T(m) = S(m+1) - S(m);       %interexceedance times
37          end
38
39          bootTheta_int = zeros(50,1);
40          bootTheta_runs1 = zeros(50,1);
41          bootTheta_runs5 = zeros(50,1);
42          bootTheta_runs9 = zeros(50,1);
43
44          for j=1:50
45              [NewT,NewN] = bootstrapmySeries_Working(T);
46              bootTheta_int(j) = getIntervalsEst(NewT);       %generate vector
                    of estimates
47              bootTheta_runs1(j) = getRunsestimator(1,NewN,NewT);
48              bootTheta_runs5(j) = getRunsestimator(5,NewN,NewT);
49              bootTheta_runs9(j) = getRunsestimator(9,NewN,NewT);
50          end
51
```

```
52          lowerTemp_int = mean(bootTheta_int) - 1.96*std(bootTheta_int);      %
                make intervals
53          upperTemp_int = mean(bootTheta_int) + 1.96*std(bootTheta_int);
54          if lowerTemp_int <= theta && theta <= upperTemp_int        %check
                whether true value in interval
55              amountyes_int = amountyes_int + 1;
56          end
57
58          lowerTemp_runs1 = mean(bootTheta_runs1) - 1.96*std(bootTheta_runs1);
59          upperTemp_runs1 = mean(bootTheta_runs1) + 1.96*std(bootTheta_runs1);
60          if lowerTemp_runs1 <= theta && theta <= upperTemp_runs1
61              amountyes_runs1 = amountyes_runs1 + 1;
62          end
63
64          lowerTemp_runs5 = mean(bootTheta_runs5) - 1.96*std(bootTheta_runs5);
65          upperTemp_runs5 = mean(bootTheta_runs5) + 1.96*std(bootTheta_runs5);
66          if lowerTemp_runs5 <= theta && theta <= upperTemp_runs5
67              amountyes_runs5 = amountyes_runs5 + 1;
68          end
69
70          lowerTemp_runs9 = mean(bootTheta_runs9) - 1.96*std(bootTheta_runs9);
71          upperTemp_runs9 = mean(bootTheta_runs9) + 1.96*std(bootTheta_runs9);
72          if lowerTemp_runs9 <= theta && theta <= upperTemp_runs9
73              amountyes_runs9 = amountyes_runs9 + 1;
74          end
75      end
76
77      Coverage_int(Q) = amountyes_int/100;          %calculate probability
78      Coverage_runs1(Q) = amountyes_runs1/100;
79      Coverage_runs5(Q) = amountyes_runs5/100;
80      Coverage_runs9(Q) = amountyes_runs9/100;
81  end
82
83  % Coverage_int
84  % Coverage_runs1
85  % Coverage_runs5
```

```
% Coverage_runs9

plot(xaxis,Coverage_int, 'k-')          %plot figures
hold on
plot(xaxis,Coverage_runs1, 'k:')
hold on
plot(xaxis,Coverage_runs5, 'k-.')
hold on
plot(xaxis,Coverage_runs9, 'k--')
end
```

```
function powerGraph(N)
% POWERGRAPH generates Figure 2
%
% N is the amount of exceedances used (determines whether we have 2(a),
% 2(b), 2(c) or 2(d)

xaxis = zeros(20,1);          %initialize values
powervector = zeros(20,1);
powervectorruns = zeros(20,1);
zerofive = zeros(20,1);
%rho = 0.6;                    %fix other parameters for 'true' value of
                               %statistic
%eta = -0.9;
for n=1:20                     %take steps of 0.5 for c
    c=(n/2)+0.5;
    xaxis(n) = c;
%     [Spos, Sneg, Tpos, Tneg] = getST(N,data);
%      p = 1/(1+c^2);
%      q = 1-p;
    rejected = zeros(100,1);
    rejectedruns = zeros(100,1);

%     Calculate Theoretical Value of the Statistic:


```

```
26 %     theta_plus = (p+q*(abs(rho+eta)^2))/(p+q*(abs(rho+eta)^2)/(1-(rho^2)));
27 %     theta_minus = (q+p*(abs(rho+eta)^2))/(q+p*(abs(rho+eta)^2)/(1-(rho^2)));
28 %     Statistic_theor = theta_plus-theta_minus
29 %     Statistic_est = getRunsestimator(5,Tpos) - getRunsestimator(5,Tneg);
30 %     Statistic_estint = getIntervalsEst(Tpos) - getIntervalsEst(Tneg);
31 %     xaxis(n) = Statistic_theor;
32
33     for j=1:100
34         data = Simulatefromskewedt(c);       %simulate new data
35         Statistic = zeros(50,1);
36         Statisticruns = zeros(50,1);
37         for i=1:50                           %calculate statistic for the runs
               estimator and intervals estimator
38           newdata = stationaryBootstrap(data);
39           [Snewpos, Snewneg, Tnewpos, Tnewneg] = getST(N,newdata);
40           thetapos = getIntervalsEst(Tnewpos);
41           thetaneg = getIntervalsEst(Tnewneg);
42           newdata = stationaryBootstrap(data);
43           [Snewpos, Snewneg, Tnewpos, Tnewneg] = getST(N,newdata);
44           thetaposruns = getRunsestimator(5,Tnewpos);
45           thetanegruns = getRunsestimator(5,Tnewneg);
46             Statistic(i) = thetapos - thetaneg;
47             Statisticruns(i) = thetaposruns - thetanegruns;
48         end
49         upper = mean(Statistic)+1.96*std(Statistic);        %get confidence
               interval
50         lower = mean(Statistic)-1.96*std(Statistic);
51         if upper > 0 && lower < 0
52             continue
53         else
54             rejected(j) = rejected(j)+1;                    %add to rejected
                   if outside interval
55         end
56         upperruns = mean(Statisticruns)+1.96*std(Statisticruns); %get
               confidence interval for runs estimator
57         lowerruns = mean(Statisticruns)-1.96*std(Statisticruns);
```

```
58          if upperruns > 0 && lowerruns < 0
59              continue
60          else
61              rejectedruns(j) = rejectedruns(j)+1; %add to rejected if outside
                    interval for runs estimator
62          end
63      end
64      powervector(n) = mean(rejected); %get fraction of rejected
65      powervectorruns(n) = mean(rejectedruns);
66      zerofive(n) = 0.05; %get horizontal line at 5%
67  end
68
69
70  plot(xaxis,powervector, 'k:') %plot figure
71  hold on
72  plot(xaxis,powervectorruns, 'k--')
73  hold on
74  plot(xaxis, zerofive, 'k-')
75  xlim([1,10])
76  ylim([0,1])
77  end
```

```
1   function figure3a_working(data)
2   % FIGURE3A_WORKING plots figure 3(a)
3   %
4   % data is the temperature data in Wooster, Ohio
5
6   extremal = zeros(15,1); %initialize values
7   lower = zeros(15,1);
8   upper = zeros(15,1);
9   xaxis = zeros(15,1);
10
11  for threshold = 1:15 %calculate for 15 different thresholds
12      xaxis(threshold) = threshold-0.5;
13      S = zeros(length(data),1);
14      exceedancetime = 1;
```

```
    for q = 1:length(data)
        if data(q) < (-threshold+0.5)
            S(exceedancetime) = q;
            exceedancetime = exceedancetime+1;
        end
    end
    Sreal = S(find(S~=0));
    length(Sreal)
    T = zeros(length(Sreal)-1,1);
    for m = 1:length(Sreal)-1
        T(m) = S(m+1) - S(m);
    end
    extremal(threshold) = getIntervalsEst(T); %calculate intervals estimator
    ReEstTheta = zeros(20,1);
    for i=1:200
        [NewT, NewN] = bootstrapmySeries(T);
        ReEstTheta(i) = getIntervalsEst(NewT); %calculate intervals estimator
            200 times
    end
    lower(threshold) = mean(ReEstTheta) - 1.96*std(ReEstTheta); %get
        confidence interval
    upper(threshold) = mean(ReEstTheta) + 1.96*std(ReEstTheta);

end
plot(xaxis,extremal, 'k-') %plot figure
hold on
plot(xaxis,lower, 'k:')
hold on
plot(xaxis,upper, 'k:')

end
```

```
function figure3b_working(data)
%FIGURE3B_WORKING plots the runs estimator for different thresholds (figure
%3b)
%
```

```
% data is the Temperature data

extremal = zeros(15,1);     %initialize values
lower = zeros(15,1);
upper = zeros(15,1);
xaxis = zeros(15,1);

for threshold = 1:15        %calculate runs estimator for 15 thresholds
    xaxis(threshold) = threshold-0.5;
    S = zeros(length(data),1);
    exceedancetime = 1;
    for q = 1:length(data)
        if data(q) < (-threshold+0.5)       %get exceedance times
            S(exceedancetime) = q;
            exceedancetime = exceedancetime+1;
        end
    end
    Sreal = S(find(S~=0));
    T = zeros(length(Sreal)-1,1);
    for m = 1:length(Sreal)-1           %get interexceedance times
        T(m) = S(m+1) - S(m);
    end
    extremal(threshold) = getRunsestimator(5,length(Sreal),T);  %get runs
        estimator
    ReEstTheta = zeros(20,1);
    for i=1:100         %get runs estimator for 100 bootstrapped series
        [NewT, NewN] = bootstrapmySeries(T);
        ReEstTheta(i) = getRunsestimator(5,NewN,NewT);
    end
    lower(threshold) = mean(ReEstTheta) - 1.96*std(ReEstTheta);    %get
        confidence interval
    upper(threshold) = mean(ReEstTheta) + 1.96*std(ReEstTheta);

end
plot(xaxis,extremal, 'k-') %plot figure
hold on
```

```
plot(xaxis,lower, 'k:')
hold on
plot(xaxis,upper, 'k:')
ylim([0.2,1])
end
```

```
function figure3c_working(data)
%FIGURE3C_WORKING plots the mean cluster excess (figure 3(c))
%
% data is the Temperature data

meanclusterexcess = zeros(15,1); %initialize values
minVal = zeros(15,1);
maxVal = zeros(15,1);
xaxis = zeros(15,1);
for threshold=1:15 %calculate mean cluster excess for 15 different thresholds
    xaxis(threshold) = threshold-0.5;
    %threshold = getThreshold(exceedances,data);
    S = zeros(length(data),1);
    exceedancetime = 1;
    for q = 1:length(data)
        if data(q) < (-threshold+0.5)        %get exceedances
            S(exceedancetime) = q;
            exceedancetime = exceedancetime+1;
        end
    end
    Sreal = S(find(S~=0));
    T = zeros(length(Sreal)-1,1);
    for m = 1:length(Sreal)-1
        T(m) = S(m+1) - S(m);              %get interexceedance times
    end
    C = floor(getIntervalsEst(T)*(length(T)+1)) + 1;
    SortedTimes = [sort(T,'descend');zeros(1000,1)];
    while SortedTimes(C-1) == SortedTimes(C)
        C = C-1;
    end
```

```
31      interClusterT = maxk(T,C-1);
32      smallest_inter = min(interClusterT);
33      clusterexcess = zeros(length(T),1);
34      c=1;
35      for i = 2:length(T)              %add total clusterexcess iteratively
36          if T(i) >= smallest_inter
37              c = c+1;
38          end
39          clusterexcess(c) = clusterexcess(c) + ((-threshold+0.5)-data(S(i-1)));
40      end
41      cleanedExcess = zeros(c,1);
42      for t=1:c
43          cleanedExcess(t,1) = clusterexcess(t);  %get only nonzero values
44      end
45
46      meanbootstrappedExcess = zeros(50,1);
47      for y = 1:50                         %calculate mean cluster excess for new
            data (bootstrapped from data)
48          bootstrappedExcess = zeros(c,1);
49          for b = 1:c
50              select = ceil(rand(1)*c);
51              bootstrappedExcess(b) = cleanedExcess(select,1);
52          end
53          meanbootstrappedExcess(y) = mean(bootstrappedExcess);
54      end
55      minVal(threshold) = mean(meanbootstrappedExcess) - 1.96*std(
            meanbootstrappedExcess); %get confidence interval
56      maxVal(threshold) = mean(meanbootstrappedExcess) + 1.96*std(
            meanbootstrappedExcess);
57      meanclusterexcess(threshold) = mean(cleanedExcess);
58  end
59  plot(xaxis,meanclusterexcess, 'k-')      %plot figure
60  hold on
61  plot(xaxis,minVal, 'k:')
62  hold on
63  plot(xaxis,maxVal, 'k:')
```

```
64 end

1  function figure3d(data)
2  %FIGURE3D plots the run length for different thresholds
3  %
4  % data is the Temperature data
5
6  runsestimator = zeros(15,1);    %initialize values
7  lower = zeros(15,1);
8  upper = zeros(15,1);
9  xaxis = zeros(15,1);
10
11 for threshold = 1:15          %calculate run length for different thresholds
12     xaxis(threshold) = threshold-0.5;
13     S = zeros(length(data),1);
14     exceedancetime = 1;
15     for q = 1:length(data)
16         if data(q) < (-threshold+0.5)   %get exceedance times
17             S(exceedancetime) = q;
18             exceedancetime = exceedancetime+1;
19         end
20     end
21     Sreal = S(find(S~=0));
22     T = zeros(length(Sreal)-1,1);
23     for m = 1:length(Sreal)-1           %get interexceedance times
24         T(m) = S(m+1) - S(m);
25     end
26     C = ceil(getIntervalsEst(T)*(length(T)+1));
27     SortedTimes = [sort(T,'descend');zeros(1000,1)];
28     while SortedTimes(C-1) == SortedTimes(C)
29         C = C-1;
30     end
31     interClusterT = maxk(T,C-1);
32     smallest_inter = min(interClusterT);
33     runsestimator(threshold) = smallest_inter;      %get run length for data
34     bootstrappedRuns = zeros(100,1);
```

```
35      for b = 1:100                                   %get run length for 100
            bootstrapped datasets
36          [NewT, NewN] = bootstrapmySeries_Working(T);
37          C = ceil(getIntervalsEst(NewT)*NewN);
38          SortedTimes = [sort(NewT,'descend');zeros(1000,1)];
39          while SortedTimes(C-1) == SortedTimes(C)
40              C = C-1;
41          end
42          interClusterT = maxk(NewT,C-1);
43          smallest_inter = min(interClusterT);
44          bootstrappedRuns(b) = smallest_inter;
45      end
46      lower(threshold) = max(0,mean(bootstrappedRuns) - 1.96*std(bootstrappedRuns
            )); %get confidence interval
47      upper(threshold) = mean(bootstrappedRuns) + 1.96*std(bootstrappedRuns);
48  end
49  plot(xaxis,runsestimator, 'k-') %plot figure
50  hold on
51  plot(xaxis,lower, 'k:')
52  hold on
53  plot(xaxis,upper, 'k:')
54  ylim([-5,50])
55  end
```

```
1   function [Psimes] = plotNewStatistic(data)
2   %PLOTNEWSTATISTIC plots figures 4, 5(c) and 5(d)
3   %
4   % data is the input data (either Temperature or stock returns)
5
6   n = length(data);       %initialize all values
7   replications = 50;
8   lowerbound = zeros(50,1);
9   upperbound = zeros(50,1);
10  values = zeros(50,1);
11  xaxis = zeros(50,1);
12  Pvalue = zeros(50,1);
```

```
PsimesVec = zeros(50,1);
Psimes = 0;
originalvalues = zeros(50,1);

for Q = 1:50          %calculate statistic for 50 different thresholds
    N = Q*10;
    xaxis(Q) = N;
    Statistic = zeros(replications,1);
    [Spos, Sneg, Tpos, Tneg] = getST(N,data);
    for i=1:replications
        newdata = stationaryBootstrap(data);
        [Snewpos, Snewneg, Tnewpos, Tnewneg] = getST(N,newdata);
%         thetapos = getIntervalsEst(Tnewpos);
%         thetaneg = getIntervalsEst(Tnewneg);
        thetapos = getRunsestimator(5,Tnewpos);
        thetaneg = getRunsestimator(5,Tnewneg);
        Statistic(i) = thetapos - thetaneg; %calculate the statistic many
            times
    end
    Statistic=(Statistic(~isnan(Statistic)));
    lowerbound(Q) = mean(Statistic) - 1.96*std(Statistic); %calculate interval
    upperbound(Q) = mean(Statistic) + 1.96*std(Statistic);
    z = mean(Statistic)/std(Statistic);
    Pvalue(Q) = exp(-0.717*(z)-0.416*z^2);
    PsimesVec(Q) = 50*Pvalue(Q)/Q;
    Psimes = min(PsimesVec);            %calculate the Simes P value
    values(Q) = mean(Statistic); %calculate mean of the statistic
    originalvalues(Q) = getIntervalsEst(Tpos)-getIntervalsEst(Tneg);
%     negthetas(Q) = getIntervalsEst(Tneg);
%     posthetas(Q) = getIntervalsEst(Tpos);
end
% plot(xaxis,negthetas, 'k:')
% hold on
% plot(xaxis,posthetas, 'k-')
plot(xaxis,lowerbound, 'k:')
hold on
```

```
plot(xaxis,values, 'k-')
hold on
plot(xaxis,upperbound, 'k:')
hold on
plot(xaxis,zeros(50,1),'k-')
ylim([-1,1])
end
```

```
function figure5abSP(data)
% FIGURE5ASP plots figure 5a and 5b with slight alteration (change
% intervals estimator to runs estimator)
%
% data is the S&P500 daily log returns

extremal = zeros(25,1); %initialize values
lower = zeros(25,1);
upper = zeros(25,1);
xaxis = zeros(25,1);

for amount = 1:25        %get intervals estimator for 25 different thresholds
    xaxis(amount) = amount*20;
    [Spos, Sneg, Tpos, Tneg] = getST(xaxis(amount), data); %get T positive
    extremal(amount) = getIntervalsEst(Tpos);
    ReEstTheta = zeros(50,1);
    for i=1:50
        [NewT, NewN] = bootstrapmySeriesSP(Tpos);      %bootstrap from the
            real interexceedance times
        ReEstTheta(i) = getIntervalsEst(NewT);      %re-estimate the intervals
             estimator for bootstrapped data
    end
    lower(amount) = mean(ReEstTheta) - 1.96*std(ReEstTheta); %get confidence
        interval
    upper(amount) = mean(ReEstTheta) + 1.96*std(ReEstTheta);

end
plot(xaxis,extremal, 'k-') %plot the figure
```

```
hold on
plot(xaxis,lower, 'k:')
hold on
plot(xaxis,upper, 'k:')
ylim([0,1])
end

function [NewT, NewN] = bootstrapmySeries_Working(T)
%BOOTSTRAPMYSERIES_WORKING bootstraps the interexceedances times and returns the
%new amount of exceedances
%
% T are the initial interexceedance times

N = length(T)+1;
C = floor(getIntervalsEst(T)*N) + 1;

SortedTimes = [sort(T,'descend'); zeros(1000,1)]; %sort the interexceedance times
while SortedTimes(C-1) == SortedTimes(C)
    C = C-1;
end
interClusterT = maxk(T,C-1);
smallest_inter = min(interClusterT);
bootstrapInter = zeros(C-1,1);      %initialize interexceedance times vector
bootstrapIntra = zeros(C,N-1);
opencluster = true;
c = 1;
x = 1;
largeT = 1;
amountofEmpty = 0;

for i = 1:N-1
    if T(i) < smallest_inter    %add to intracluster or start a cluster
       if opencluster == false  %if there is no open cluster
           c = c+1;
```

```
28              x = 1;
29          end
30          bootstrapIntra(c,x) = T(i);
31          x = x + 1;
32          opencluster = true;
33      else                            %add to intercluster times
34          bootstrapInter(largeT) = T(i);
35          largeT = largeT + 1;
36          if i >= 2
37              if T(i-1) >= smallest_inter
38                  amountofEmpty = amountofEmpty + 1;
39              end
40          end
41          opencluster = false;
42      end
43  end
44  bootstrapIntra;
45  T;
46
47  NewInter = zeros(2*C,1);
48  for i = 1:(C-1)      %make vector of interexceedance times only
49      select = ceil(rand(1)*(C-1));
50      NewInter(i) = bootstrapInter(select);
51  end
52  NewInter = NewInter(find(NewInter~=0));
53  testlength = length(NewInter);
54  C;
55  w = 1;
56  j=1;
57  NewT = zeros(2*N,1);
58  testtt = 2*C-1;
59  for i=1:(2*C-1)      %here, we start the real bootstrapping
60      if mod(i,2) == 1
61          if rand(1) > (amountofEmpty/C)
62              select2 = ceil(rand(1)*C);
63              SelectedCluster = bootstrapIntra(select2, :);
```

```
64              for i=1:100          %select random cluster but continue if only
                    zero clusters
65                  if SelectedCluster(1) == 0
66                  select3 = ceil(rand(1)*C);
67                  SelectedCluster = bootstrapIntra(select3, :);
68                  end
69              end
70              SelectedClusterClean = SelectedCluster(find(SelectedCluster~=0));
71                  for u=1:length(SelectedClusterClean)
72                      NewT(w) = SelectedClusterClean(u);
73                      w = w + 1;
74                  end
75          end
76      else
77          NewT(w) = NewInter(j);
78          j = j + 1;
79          w = w + 1;
80      end
81  end
82  NewT = NewT(find(NewT~=0));
83  length(NewT);
84  NewN = length(NewT)+1;
85
86
87  end
```

```
1   function [intervals_est] = getIntervalsEst(T)
2   %GETINTERVALSEST estimates intervals estimate for a given set of
        interexceedance
3   %times
4   %
5   % T is the set of interexceedance times
6
7   N = length(T)+1;     %initialize values
8   SumTi = 0;
9   SumTisq = 0;
```

```
SumTimin1 = 0;
SumTimin12 = 0;

if max(T) <= 2       %apply the formula for the intervals estimator
    for i = 1:N-1
        SumTi = SumTi+T(i);
        SumTisq =SumTisq + T(i)^2;
    end
    intervals_est = (2*(SumTi^2))/((N-1)*SumTisq);
else
    for i = 1:N-1
        SumTimin1 = SumTimin1 + (T(i)-1);
        SumTimin12 = SumTimin12 + (T(i)-1)*(T(i)-2);
    end
    intervals_est = (2*(SumTimin1^2))/((N-1)*SumTimin12);
end
end
```

```
function [runsestimator] = getRunsestimator(runlength, T)
%GETRUNSESTIMATOR returns the runs estimator for given runlength and
%interexceedance times
%
% runlength is the run length
% T is a vector of interexceedance times

sum = 0;
N = length(T)+1;
for i = 1:N-1           %apply the formula for the runs estimator
    if T(i) > runlength
        sum = sum + 1;
    end
end

runsestimator = (1 + sum)/N;

end
```

```
function [Spos, Sneg, Tpos, Tneg] = getST(N, data)
%GETST returns the exceedance times and interexceedance times of both tails
%
% N is the amount of exceedances desired
% data is the data used

B = sort(data,'descend');
thresholdValuepos = B(N)-0.0001; %get positive threshold

C = sort(data,'ascend');
thresholdValueneg = C(N)+0.0001; %get negative threshold

Spos = zeros(N,1);
exceedancetime = 1;
for q = 1:length(data)  %get exceedance times for positive extremes
    if data(q) > thresholdValuepos
        Spos(exceedancetime) = q;
        exceedancetime = exceedancetime+1;
    end
end
Spos = Spos(find(Spos~=0));
Ntemp = length(Spos);
Tpos = zeros(Ntemp-1,1);
for m = 1:Ntemp-1       %get interexceedance times for positive extremes
    Tpos(m) = Spos(m+1) - Spos(m);
end

Sneg = zeros(N,1);
exceedancetime = 1;
for e = 1:length(data)      %get exceedance times for negative extremes
    if data(e) < thresholdValueneg
        Sneg(exceedancetime) = e;
        exceedancetime = exceedancetime+1;
    end
end

```

```
Tneg = zeros(N-1,1);
for m = 1:N-1            %get interexceedance times for negative extremes
    Tneg(m) = Sneg(m+1) - Sneg(m);
end

end
```

```
function [threshold] = getThreshold(N, sequence)
%GETTHRESHOLD returns the threshold hight for each N
%
% N is the amount of exceedances wanted
% sequence is the data input

B = sort(sequence,'ascend'); %sort observations
B
threshold = B(N)-0.0001; %get the threshold

end
```

```
function [xisequence] = getXi(theta)
%GETXI Generates one Xi sequence
%
% theta is the 'true' extremal index

xisequence = zeros(5000,1);
n = 5000;
xisequence(1) = (-log(rand(1)))^(-1)/theta; %start the sequence
for i =2:n                      %iterate the sequence
    xisequence(i) = max((1-theta)*xisequence(i-1),(-log(rand(1)))^(-1));
end

end
```

```
function [data] = Simulatefromskewedt(c)
% SIMULATEFROMSKEWEDT returns a dataset from a potentially skewed t
% distribution with the ARMA(1,1) process
%
```

```
% c is the parameter 'c' in the simulation studies

data = zeros(2000,1); %initialize parameters
nu = 2;                %fix other parameters
rho = 0.6;
eta = -0.9;
data(1) = trnd(nu);
shocks = zeros(2000,1);
shocks(1) = data(1);

for i=2:length(data)
    shocks(i) = trnd(nu);
    if shocks(i) < 0
        shocks(i) = shocks(i)*c;       %multiply left tail by c
    end
    data(i) = rho*data(i-1)+shocks(i)+eta*shocks(i-1);  %apply ARMA(1,1) model
        iteratively
end
end
```

```
function [bootstrappeddata] = stationaryBootstrap(data)
%STATIONARYBOOTSTRAP returns a series which is a result of bootstrapping
%from the real data
%
% data is the real data

blocklength = 80;       %chosen blocklength
Q = 1-(1/blocklength);
N = length(data);
bootstrappeddata = zeros(2*N,1);
t = ceil(rand(1)*N);
bootstrappeddata(1) = data(t);
%c=1;
%blocklength = zeros(N,1);
for i=2:N           %get new series of exactly length N
    select = rand(1);
```

```
17      if select < Q       %apply bootstrap procedure
18          if t == N
19              t=1;
20          else
21              t = t+1;
22          end
23          bootstrappeddata(i) = data(t);
24          %blocklength(c) = blocklength(c)+1;
25      else
26          t = ceil(rand(1)*N);
27          bootstrappeddata(i) = data(t);
28          %blocklength(c) = blocklength(c) + 0.00001;
29          %c = c+1;
30      end
31  end
32  %blocklength = blocklength(find(blocklength~=0));
33  %mean(blocklength)
34  bootstrappeddata = bootstrappeddata(find(bootstrappeddata~=0));
35  end
```