CHALMERS | GÖTEBORG UNIVERSITY

MASTER'S THESIS

Bootstrap of Dependent Data in Finance

ANDREAS SUNESSON

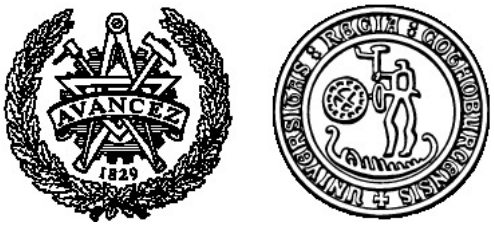
Department of Mathematical Statistics
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG UNIVERSITY
Göteborg, Sweden 2011

Thesis for the Degree of Master of Science

Bootstrap of Dependent Data in Finance

Andreas Sunesson

CHALMERS | GÖTEBORG UNIVERSITY



Department of Mathematical Statistics
Chalmers University of Technology and Göteborg University
SE – 412 96 Göteborg, Sweden
Göteborg, August 2011

Bootstrap of Dependent Data in Finance

Andreas Sunesson

August 17, 2011

Abstract

In this thesis the block bootstrap method is used to generate resamples of time series for Value–at–Risk calculation. A parameter to the method is the block length and this parameter is studied to see what choice of it gives the best results when looking at multi–day Value–at–Risk estimates. A choice of block length of the roughly the same size as the number of days used for Value–at–Risk seems to give the best results.

Acknowledgements

I would like to thank my supervisor, at Chalmers, Patrik Albin for his support during my work. He his an ideal supervisor for a student. Further on i wish to thank Magnus Hellgren and Jesper Tidblom and the rest of Algorithmica Research AB for their support.

Contents

1 Introduction

In financial institutions there is an ever increasing need to estimate the risk of various positions. Today there exists a number of measures available to risk managers. A risk measure made popular by J.P. Morgan is the Value–at–Risk measure, which simply is the a chosen quantile of the loss distribution. Value–at–Risk quickly became popular in the industry was also soon adopted by the Basel Committee on Banking Supervision who regulated that a certain amount of capital was needed to cover for the risk exposure indicated by the Value–at–Risk measure. Initially, however, Value–at–Risk was based on an assumption of independent normal distributed asset returns, mainly because of the simplicity of calculation and the ease of explaining to senior managment. As known though, real world assets, in general, are neither normal distributed nor independent.

This way of estimating Value–at–Risk is by the use of a parametric method, assumptions has been made of the underlying asset return process and parameters has been estimated from historical data. By the use of a non–parametric model, though, one can drop a lot of these model assumptions. One such non–parametric model is resampling by the Bootstrap method, initially proposed by Efron in 1979 [3]. This non–parametric model assymptotically replicates the density of the resampled process. The bootstrap method assumes independent asset returns and a problem with it, if you try to apply it on a dependent time series, is that the resampled series is independent. To correct for this some modifications to the bootstrap method was later proposed.

In this thesis, dependent time series will be used to study extended versions of the bootstrap method, the block bootstrap and the stationary bootstrap. The choice of parameters for the methods are of particular interest and are studied for empirical data by different approaches. Also, how does resampling by these methods preserve the autocorrelation structure in the resamples and further on Value–at–Risk estimates will be made based on the resamples.

2 Theory

2.1 Basic definitions

We begin by looking at an asset with a price process $\{S_t\}_{t=0}^T$, $S_t > 0$. We're now interested in the returns of this price process. There are a couple of ways to define this, starting with the *simple net return*, R_t defined by

$$R_t = \frac{S_t}{S_{t-1}} - 1$$

Furtheron, define the *log return*, X_t, $t \in [1, T]$ by

$$\begin{aligned} X_t &= \log(1 + R_t) \\ &= \log\left(\frac{S_t}{S_{t-1}}\right). \end{aligned}$$

If we look at multiperiod returns $R_t(\ell)$ and $X_t(\ell)$ we see that they are obtained by multiplying the daily simple net returns between the desired days and adding the log returns for the same period.

Further on we need to define the bias and the mean square error of an estimate. Suppose that $\{X_t\}_{t=0}^T$ is identically distributed with some common distribution function, G. Suppose also that in an experiment we get a sample from the previous sequence, $\{X_1 \dots X_n\}$. Now suppose we're interested in the functional relation $\theta = \theta(G)$ but don't know the distribution G. We might estimate G with $\widetilde{G}$, from the sample $\{X_1 \dots X_n\}$. That is, $\widetilde{G}(X_1, \dots, X_n)$. Thus an estimate of θ is gotten by applying $\widetilde{G}$ instead of G. Now is this estimate, $\widehat{\theta} = \theta(\widetilde{G})$ a good one? To check this we introduce the bias,

$$\text{Bias}(\widehat{\theta}) = \mathbb{E}\left[\widehat{\theta}\right] - \theta,$$

which is essentially the average error we make by estimating θ with $\widehat{\theta}$.

Also we introduce the mean square error and define it by

$$\begin{aligned} \text{MSE}(\widehat{\theta}) &= \mathbb{E}\left[\left(\widehat{\theta} - \theta\right)^2\right] \\ &= \text{Bias}(\widehat{\theta})^2 + \text{Var}(\widehat{\theta}). \end{aligned}$$

2.2 Autocorrelation function

The correlation, between two random variables X and Y, both square integrable, is defined through

$$\rho(X, Y) = \text{Cov}(X, Y)/\sqrt{\text{Cov}(X, X)\text{Cov}(Y, Y)}.$$

The sample correlation between two samples $X = \{X_1, \dots, X_n\}$ and $Y = \{Y_1, \dots, Y_n\}$ with $\bar{X} = \frac{1}{n}\sum_{i=1}^n X_i$ and $\bar{Y} = \frac{1}{n}\sum_{i=1}^n Y_i$ is defined by

$$\rho(X, Y) = \frac{\sum_{k=0}^n (X_k - \bar{X})(Y_k - \bar{Y})}{\sqrt{\sum_{k=0}^n (X_k - \bar{X})^2 \sum_{k=0}^n (Y_k - \bar{Y})^2}}.$$

For a sample $X = \{X_1 \ldots, X_n\}$ it's correlation with it self, for a lag ℓ, is defined as the auto-correlation function, "acf", [12].

$$\rho_\ell = \frac{\sum_{i=\ell+1}^{T} \left(X_i - \bar{X}\right)\left(X_{i-\ell} - \bar{X}\right)}{\sum_{i=1}^{T} \left(X_i - \bar{X}\right)^2}, \ell \in [0, T-1]$$

Here a value of 1 for a specific ℓ would indicate that there is a perfect correlation between the series starting at day k and the series starting at day $k + \ell$. Note that $-1 \leq \rho \leq 1$.

2.3 Serial variance

The sample variance for $X = \{X_1, \ldots, X_n\}$ is calculated by

$$\widehat{\mathrm{Var}}(X) = \frac{1}{n-1}\left(\sum_{i=1}^{n} X_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} X_i\right)^2\right).$$

The reason for dividing by $n-1$ instead of n is that this makes $\mathrm{Var}(X)$ an unbiased estimator of the variance if the process X is IID and stationary.

When looking at multiperiod returns instead, how do we expect the sample variance to change? If we look at k–day changes, the sample size will decrease by a factor k. Thus we expect to see an increase in the sample variance since it is expected to increase with a decreasing sample.

Now, looking at k–day, non–overlapping, returns, $X_i[k]$. Assume that they are stationary and IID and that k divides n evenly. The data set is $Y = \{Y_1, \ldots, Y_{n/k}\}$, where $Y_i = X_{i(k-1)+1} + \cdots + X_{ik}$ for $i = \{1, \ldots, n/k\}$. We thus have n/k number of observations, the sample variance of this sample is, with an index k indicating the multiperiod returns,

$$\begin{aligned}
\widehat{\mathrm{Var}}_k(X) &= \widehat{\mathrm{Var}}(Y) \\
&= \frac{1}{n/k-1}\left(\sum_{i=1}^{n/k} Y_i^2 - \frac{1}{n/k}\left(\sum_{i=1}^{n/k} Y_i\right)^2\right) \\
&= \frac{1}{n/k-1}\left(\sum_{i=1}^{n/k} \left(X_{(i-1)k+1} + \cdots + X_{ik}\right)^2 - \frac{1}{n/k}\left(\sum_{i=1}^{n/k} \left(X_{(i-1)k+1} + \cdots + X_{ik}\right)\right)^2\right).
\end{aligned}$$

Now assume the data is IID and stationary,

$$\begin{aligned}
\mathbb{E}\left[\mathrm{Var}_k(X)\right] &= \mathbb{E}\left[\frac{1}{n/k-1}\left(\sum_{i=1}^{n/k}\left(X_{(i-1)k+1}+\cdots+X_{ik}\right)^2-\frac{1}{n/k}\left(\sum_{i=1}^{n/k}\left(X_{(i-1)k+1}+\cdots+X_{ik}\right)\right)^2\right)\right] \\
&= \frac{k}{n-k}\left(\sum_{i=1}^{n/k}\mathbb{E}\left[\left(X_{(i-1)k+1}+\cdots+X_{ik}\right)^2\right]-\frac{k}{n}\mathbb{E}\left[\left(\sum_{i=1}^{n}X_i\right)^2\right]\right) \\
&= \frac{k}{n-k}\left(\sum_{i=1}^{n/k}\left(\mathrm{Var}\left(X_{(i-1)k+1}+\cdots+X_{ik}\right)+\mathbb{E}\left[X_{(i-1)k+1}+\cdots+X_{ik}\right]^2\right)\right. \\
&\quad \left.-\frac{k}{n}\left(\mathrm{Var}\left(\sum_{i=1}^{n}X_i\right)+\mathbb{E}\left[\sum_{i=1}^{n}X_i\right]^2\right)\right) \\
&= \frac{k}{n-k}\left(\frac{n}{k}\left(\mathrm{Var}(X_i)\cdot k+k^2\mathbb{E}\left[X_i\right]^2\right)-\frac{k}{n}\left(n\cdot\mathrm{Var}\left(X_i\right)+n^2\mathbb{E}[X_i]^2\right)\right) \\
&= \frac{k}{n-k}\left(n\cdot\mathrm{Var}\left(X_i\right)+nk\mathbb{E}\left[X_i\right]^2-k\cdot\mathrm{Var}\left(X_i\right)-nk\mathbb{E}\left[X_i\right]^2\right) \\
&= k\cdot\mathrm{Var}(X_1),
\end{aligned}$$

we see that the sample variance increases by a factor of k when the sample size decreases by a factor $1/k$.

Now if a sample does not show this tendency, apart from random noise, we could be inclined to say that the sample is not independent.

2.4 QQ–plots

Assume a stationary sample $\{X_1,\ldots,X_n\}$. A quantile–quantile plot, qq–plot, is a visual way of checking if the data in the sample belongs to a specified distribution. The ordered sample, $\{X_{(1)},\ldots,X_{(n)}\}$, where $X_{(i)}$ is the i:th largest element, is plotted against the quantiles of the corresponding inverted density function.

$$\left(X_{(i)},F^{-1}\left(\frac{i-0.5}{n}\right)\right),i\in[1,n]$$

The qq–plot can also be used to check if two samples come from the same distribution. Assume that the samples are of equal size, then the corresponding qq–plot is gotten by ordering the two samples with increasing values and plotting them against each other. If both samples belong to the same distribution, the plot is expected to lie on a 45° line. If the samples are of different sizes, the same principle applies but a form of interpolation between data points is required to make sure the same quantiles are being plotted against each other.

A deviation from the 45° linear fit between the two samples would thus indicate that the two samples does not share the same distribution and it is in this manner the qq–plots will be used in this thesis. A bad fit will be "seen" as a rejection of some hypothesis of the samples not having the same distribution. This does not imply that they have the same distribution but it will indeed be an indication of it.

2.5 Kolmogorov–Smirnov test

The Kolmogorov–Smirnov test, KS–test, is a hypothesis test that can be used for testing if two samples belong to the same distribution. The KS–test usually analyzes a data set to check if it comes from a known distribution. However, in this article we are looking at non–parametric methods and do not want to mix in any assumptions of that the data has a certain distribution function. We will compare the empirical distribution function of the resampled multi–period returns against the original series empirical distribution function.

Under the null–hypothesis, the two compared data sets have the same distribution function. This is tested by calculating the statistic $D = \max_x |F_{\text{est}}(x) - F_{\text{emp}}(x)|$. Under the null–hypothesis, $\sqrt{\frac{n_1 n_2}{n_1+n_2}}D$ is Kolmogorov distributed where n_1 and n_2 are the sample sizes of the first and the second series. Thus if the largest difference between the distribution functions are too large, the null–hypothesis is rejected.

2.6 Value–at–Risk, VaR

Value–at–Risk, $\text{VaR}_\alpha(\ell)$, is the magnitude of which losses does not exceed, with probability $p = 1-\alpha$, $\alpha \in [0,1]$ in a timespan of ℓ days. That is, $\text{VaR}_\alpha(\ell)$ is defined through

$$p = \mathbb{P}\left[L(\ell) \geq \text{VaR}_\alpha\right],$$

where L is the loss function, for example $L(\ell) = -\left(S_{t+\ell} - S_t\right)$.

This can be rewritten on the form

$$\text{VaR}_\alpha(L) = \inf\left\{l \in \mathbb{R} : \mathbb{P}\left(L \leq l\right) \geq \alpha\right\}$$

Note that VaR has a *weak linearity property*, that is for $\sigma \in (0,\infty)$ and $\mu \in \mathbb{R}$,

$$\begin{aligned}\text{VaR}_\alpha\left(\sigma L + \mu\right) &= \inf\left\{l \in \mathbb{R} : \mathbb{P}\left(\sigma L + \mu \leq l\right) \geq \alpha\right\} \\ &= \inf\left\{l \in \mathbb{R} : \mathbb{P}\left(L \leq (l-\mu)/\sigma\right) \geq \alpha\right\},\end{aligned}$$

substitute $k = (l-\mu)/\sigma$,

$$\begin{aligned}&= \inf\left\{\sigma k + \mu \in \mathbb{R} : \mathbb{P}\left(L \leq k\right) \geq \alpha\right\} \\ &= \sigma \inf\left\{k \in \mathbb{R} : \mathbb{P}\left(L \leq k\right) \geq \alpha\right\} + \mu \\ &= \sigma \text{VaR}_\alpha\left(L\right) + \mu.\end{aligned}$$

The textbook way of calculating VaR makes use of this fact when it assumes normal distributed log–returns. Calculating VaR is then reduced to estimating the mean,μ, and the standard deviation,σ, of the log–returns. That is, at time t, $\text{VaR}_\alpha = S(t)(\mu + \sigma\Phi(\alpha))$.

2.7 GARCH

A way of estimating volatility, σ_t^2, in financial time series is by applying a Generalized autoregressive conditional heteroscedastic, GARCH, model. A GARCH(p,q) model for estimating the volatility

is described by

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i a_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2$$

where a_t is zero–mean adjusted log return process described in section 2.1. Also $a_t = \sigma_t \epsilon_t$ and, in this thesis, $\epsilon_t \sim N(0, 1)$.

From the way the GARCH model is constructed, at each time t, the volatility process σ_t^2 is constructed from the previous q volatilities, weighted down by the β_i–coefficients. Also the process takes the last p innovation values, a_t into consideration, weighted down by the α_i–coefficients.

To estimate the parameters $\alpha_0, \ldots, \alpha_p,\ \beta_1, \ldots, \beta_q$ from a data sample, one method commonly used is MLE.

2.8 Ljung–Box test

The Ljung–Box test is a hypothesis test for determining if the autocorrelation function is non–zero, [12].

- H_0: $\rho(1), \ldots, \rho(m)$ is all zero.
- H_1: at least one of $\rho(1), \ldots, \rho(m)$ is non–zero.

Thus under the null–hypothesis,

$$Q(m) = T\,(T+2) \sum_{\ell=1}^{m} \frac{\widehat{\rho}_\ell^2}{T-\ell}$$

is chi–square, χ_m^2, distributed with m degrees of freedom.

2.9 Kupiec–test

A Kupiec–test is a hypothesis test for checking if the number of exceptions in a backtest is as many as expected, given that an exception occurs independently of the others [5].

Let N_{obs} be the number of observed exceptions in a backtest. Let N_{exp} be the expected number of exceptions in the backtest.

- H_0: $N_{\text{obs}} = N_{\text{exp}}$
- H_1: $N_{\text{obs}} \neq N_{\text{exp}}$

That is, under the null–hypothesis N_{obs} will be binomial distributed with parameter p, where p is the level of VaR–measurement.

3 Resampling methods

The bootstrap method is a commonly used way of checking the distribution function of some estimator on a time series. Here we give a definition of the method and supply an exampel, with dependent data, where the standard method fails. We then continue to expand the method to behave descent for dependent data.

3.1 The IID Bootstrap method

The main principle of the bootstrap method is resampling from a known data set, $X_1, X_2, \ldots, X_n$, to give a distribution of the estimator $\widehat{\theta}$. This is done in the IID Bootstrap–method by picking n numbers from $\{1, \ldots, n\}$ with equal probability and with replacement. Call a number chosen this way U to get a sequence U_i, $i \in [1, n]$. Then construct a resample by chosing n data points X_{U_i}, $i \in [1, n]$, call this series $\widetilde{X_1}, \ldots, \widetilde{X_n}$ and then applying this resample to the estimator $\widehat{\theta}$. By repeating this process N times we get a series of estimations $\left\{\widehat{\theta}(\widetilde{X_1}^j, \ldots, \widetilde{X_n}^j)\right\}_{j=1}^{N}$ which can be used to construct a distribution function, $\mathbb{P}\left(\widehat{\theta} \leq x\right)$ of $\widehat{\theta}$ [1].

The IID Bootstrap–method is based on the idea that the data, $X_1, X_2, \ldots, X_n$, only represents a single realisation of all possible combinations of that data. The distribution of the estimate $\widehat{\theta}$ given by the IID Bootstrap method is hence an estimate of the distribution of $\widehat{\theta}$ if $X_1, X_2, \ldots, X_n$ came from a common distribution function $P(X \leq x)$. This however requires the data to be independent. If the data is in fact dependent, the method fails to give a proper estimate, as seen in following example, modified from an article by Eola Investments , LLC [4].

3.2 Example of IID Bootstrap shortcomings

The article proposes a gamble, you're allowed to buy as large a series as you want for \$1 per number. The series you get is comprised of consecutive 0s and 1s. For every sequence of "10101" you can show in that series, you win \$1000.

To analyze this gamble, a series of 10000 numbers is aquired
$X = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, \ldots\}$. The mean of this series lies at 0.55 which implies that the probability, p, of encountering a 1, looking anywhere in the series, is 0.55. Assuming that the data is independent of each other, the expected value of "10101's" in a 10000 number long series is thus roughly $\approx p \cdot (1-p) \cdot p \cdot (1-p) \cdot p \cdot 10000 \approx 337$. Applying the IID Bootstrap–method to this data to estimate the possibility of having an occurence of "10101" gives a 95% confidence interval of $[295.42, 335.23, 375.03]$. However, going back to the original data and actually counting the number of occurences results in a finding of only 5.

This huge discrepancy found is explained from the fact that when calculating the theoretical expected value of occurences, we assumed that the sequence was IID, obviously, this was also the case with the IID Bootstrap–method. However, the sequence was constructed by a simple algorithm that took the previous value and kept it with probability 0.7, and changed the value to either 0 or 1, both with equal probability, with probability 0.3. Clearly, this sequence is not independent, and Monte–Carlo simulations from this process give the expected number of occurences of "10101's" to be 5.89.

3.3 Block bootstrap

As seen in previous example, the IID Bootstrap–method completly fails to replicate the properties of dependent sequences. This problem is treated by the block bootstrap method.

The general idea of the method is to construct a sample of the data by splitting it into blocks and choosing, with equal probability, among them with replacement until a new series has been created. More formaly, given a sequence $X_1, \dots, X_n$, assume that the block length $\ell \in \{1, 2, \dots\}$ evenly divide n into b blocks, i.e. $n = b \cdot \ell$. Now call B_i^ℓ the block of length ℓ starting at i, $B_i^\ell = \{X_i, \dots, X_{i+\ell}\}$. The new sample will be created by drawing b blocks from $\{B_1^\ell, B_2^\ell, \dots, B_N^\ell\}$ with replacement, here $N = n - \ell + 1$ and indicates the number of available blocks. This procedure puts a lesser weight on the first and last numbers, $\{1, \dots, \ell - 1\}$ and $\{n - \ell + 1, \dots, n\}$. To correct this the data is placed end to end with itself, $\{\dots, X_n, X_1, \dots, X_n, X_1, \dots\}$, giving us n unique blocks. This is called a circular block bootstrap, [11].

By the use of the block bootstrap method, one runs the risk of destroying time stationarity properties of the time series. To correct for this, Politis and Romano, [11], introduced a random block length. When constructing a block and choosing an element of the series, draw a random number from a $U(0, 1)$ distribution, if it is greater then a number $p \in (0, 1]$ then your block is done. Otherwise include the next element as well. Thus the block length is a random variable with a geometric distribution with parameter p with expected block length $1/p$ elements long. Thus choose the parameter, p, so that you on average have an $1/p \approx \ell$ long blocks. This modification is called the stationary bootstrap since it preserves the stationarity of the series.

3.4 Example cont.

By applying the block bootstrap method with a block length of 20 and counting the number of occurences of 10101's, we get from 1000 resamples a sample mean of 8.53 with a 95% confidence interval of $[2.67, 14.38]$. Which is a much better result than with the IID bootstrap. [4]

3.5 Optimal block length

A method of calculating the optimal block length is suggested by Lahiri and is essentialy based on the jackknife method for calculating mean square error.

The Jackknife–after–bootstrap, as presented in [6],[7], is a jackknife method, where instead of deleting individual observations, overlapping blocks of observations are deleted. That is, set $I_i = \{1, \dots, N\} \cap \{i, \dots, i + m - 1\}$ where $i = 1, \dots, M$ and $M = N - m + 1$. Then the estimate of the functional $\tilde{\varphi}_n^{(i)}$, i indicates the blocks to be left out, is the functional φ_n acting on the empirical distribution function gotten by the resampled data from $\{B_{J_j}^{(i)}\}_{j=1}^b$ where J_j is a discrete uniform random variable taking its values in I_i. The jackknife–after–bootstrap estimator for the variance is then defined by

$$\widehat{\mathrm{VAR}}_{\mathrm{JAB}} = \frac{m}{N - m} \cdot \frac{1}{M} \sum_{i=1}^{M} \left(\tilde{\varphi}_n^{(i)} - \hat{\varphi}_n \right)^2 .$$

Where $\tilde{\varphi}_n^{(i)} = (N\hat{\varphi}_n - (N-m)\hat{\varphi}_n^{(i)})/m$. The bias of φ_n is estimated by

$$\tilde{B}_n = 2\ell_1 \left(\hat{\varphi}_n(\ell_1) - \hat{\varphi}_n(2\ell_1)\right)$$

where ℓ_1 is the initial guess of blocklength.

By expansion of the MSE, under some constraints, an estimation of the optimal block length is gotten by

$$\tilde{\ell} = \left(\frac{\tilde{B}_n^2 \cdot n}{\widehat{\mathrm{VAR}}_{\mathrm{JAB}}} \right)^{\frac{1}{4}}.$$

The exponent 1/4 here comes from the fact that we are looking at one–sided distributions, see for example [7] and [10].

Further on, a starting value of the block length, ℓ_1, and the block of blocks parameter m is suggested. A choice of $\ell_1 = n^{\frac{1}{4}}$ and $m = 0.1 \cdot (n\ell_1^2)^{\frac{1}{3}}$ is said to be optimal, [7].

4 Financial timeseries

The timeseries presented in this paper comes from three different categories of assets. Indexes, exchange rates and interest rates of various expiry lengths. These series are in this section presented with various properties, such as the average value and the standard deviation.

4.1 Stylized empirical facts

A stylized fact is a property shared between most financial time series. Presented below are some well–known stylized facts.

- **Squared returns got non–zero autocorrelation**
  It's a well known fact that the autocorrelations of a series of asset returns are essentially zero. However, any larger power of the returns are generally significantly non–zero. This is often understood as a form of long–horizon–dependence [2].

- **Heavy tails**
  Looking at the tails of the asset return distribution, they show a power–law tail [2]. Looking at their excess kurtosis, they are all strictly positive, that is *leptokurtic*.

- **Assymetrical distribution**
  If the asset returns are split up into two parts, first its profits and last its losses, it can be seen that there is an assymetry between these two sets. The losses tends to be larger in size then the profits. An interpretation of this is that bad news comes as a shock to the market whilst good news are, in some sense, expected and does not make the market react as much. [2]

4.2 Interest rates

In this article, two types of interest rates are used. Both are LIBOR notes, that is London Interbank Offered Rates, similar to the Swedish version, STIBOR. As seen in the name, LIBOR is the rate banks trade between each others and are usually traded close to government bonds. The rates used in this thesis consists of

- Deposit rates
  A deposit rate, short "depo rate" or just "depo", is a contract between two parties, one who wishes to borrow capital and one who wishes to lend the capital. The depo has a specified time until maturity. The rates presented in this paper as depo rates are to be interpreted as the interest rates payed by the borrower to the lender. A depo is non–tradable. [8]

- Swap rates
  A swap of interest rates are, simplified, a trade between two parties, one with a loan with a floating–rate, short–bound rate, and one with a fix rate, long–bound rate. They both wish to change from their respecitve rate to the others and doing this, they perform a swap. The rates presented in this paper as swap rates are to be interpreted as the rate offered if one wishes to pay a 3–month LIBOR note to the specified fixed rate. [9]

4.3 The dataset

The dataset is composed of different series interest rate series, shown in table 1. The data was checked for outliers and also corrected for non–trading holidays, such as 25, 30–31 of Dec. Each

of these series span over the dates 2000-01-01 to 2011-01-01 if nothing else is specified. So called *Holiday effects* are neglected in this thesis.

Name	Maturity
EUR LIBOR	3M
EUR LIBOR	1Y
USD LIBOR	3M
USD LIBOR	1Y

Table 1: Analyzed time series.

5 Implementation

The theory previously presented is used in computer programs, the code presented in appendix A. This section presents the algorithms and methods used to construct the tests.

5.1 Optimal block length

In calculations of the optimal block length by the $\widehat{\text{VAR}}_{\text{JAB}}$, Monte–Carlo–simulations are run to estimate value–at–risk for a time series. Doing so, many resamples are produced, but these are not enough, subsampling through the jackknife method are also required. This is a very expensive, computation wise, procedure. However, through the use of following theorem a reuse of resamples are allowed.

Theorem Let $J_1, \ldots, J_b$ be iid discrete uniform random variables on $\{1, \ldots, N\}$ and let $\{J_{i1}, \ldots, J_{ib}\}$ be iid discrete uniform random variables on I_i, see section 3.5, $1 \leq i \leq M$. Also, $p_i = \sum_{j=1}^{b} \mathbf{1}_{\{J_j \in I_i\}}/b$ Then $\mathbb{P}(J_1 = j_1, \ldots, J_b = j_b | p_i = 1) = \mathbb{P}(J_{i1} = j_1, \ldots, J_{ib} = j_b)$.

That is, given that one resample misses the blocks indexed $i, \ldots, i+m-1$, the probability of it having certain blocks is the same as it having the same blocks but not being able to choose from the missing blocks in the first place [7]. Thus the procedure of calculating the jackknife–after–bootstrap estimate och the variance is a matter of finding the resamples that already have a lack of the desired blocks.

First of, start by Monte–Carlo–simulating φ_n This is done by fixing a block length, ℓ, and then simulating K resamples, $\{{}_kB_1^*, \ldots, {}_kB_b^*\}$, $k = 1, \ldots, K$. These resamples are stored for future use. For each resample, take the $100(1-\alpha)\%$ worst multi–period event and call this your value–at–risk estimate. Finally take the sample mean of the estimates.

To measure $\widehat{\text{VAR}}_{\text{JAB}}$, $\tilde{\varphi}_n^{(i)}$ is calculated by choosing the resamples, already calculated in the Monte–Carlo–simulations, that lack blocks $i, \ldots, i+m-1$. That is choose the resamples where $\{{}_kB_1^*, \ldots, {}_kB_b^*\} \cap \{B_i, \ldots, B_{i+m-1}\} = \emptyset$. The suggested optimal block length is then gotten as in section 3.5.

5.2 VaR–estimaton by block bootstrap

To use the block bootstrap method to estimate the density function of the m day multiperiod returns, $m \geq 1$, from the daily return data $\{X_i\}_{i=1}^n$, we follow these simple instructions.

- Construct a set of blocks $B^\ell = \{B_1^\ell, \ldots, B_n^\ell\}$ where $B_i^\ell = \{X_i, \ldots, X_{i+\ell}\}$.
- Draw b discrete uniform random numbers from 1 to n, call them $\{U_i\}_{i=1}^b$.
- Choose b blocks from B^ℓ by taking the block with indexes U_i. That is, $B_*^\ell = \{B_{U_1}^\ell, \ldots, B_{U_b}^\ell\} = X^* = \{X_1^*, \ldots, X_n^*\}$.
- Compound the resample, $\{X_1^* + \cdots + X_m, \ldots, X_{n-m+1} + \cdots + X_n^*\}$, $m \geq 1$
- Sort X^* by size, smallest first, and choose the $n\alpha$ first element, $X_{(n\alpha)}^*$. Store this number.
- Repeat this procedure N times and after that take the sample mean of the observations. This is an estimate of the α–VaR.

5.3 VaR–estimation by stationary bootstrap

An algorithm for VaR estimation by the stationary bootstrap is presented with these step–by–step instructions.

Construct a set of numbers $U = \{U_1, \ldots, U_n\}$ by following these procedures. If at any time U got n elements, quit the algorithm.

- Set i to 1.
- Choose a number from a discrete uniform distribution over the numbers 1 to n. Store this number as U_i.
- With probability p jump to the next step and with probability $1 - p$ increase i by 1 and jump to the previous step.
- Choose U_{i+1} as $U_i + 1$, increase i by 1 and jump to the previous step.

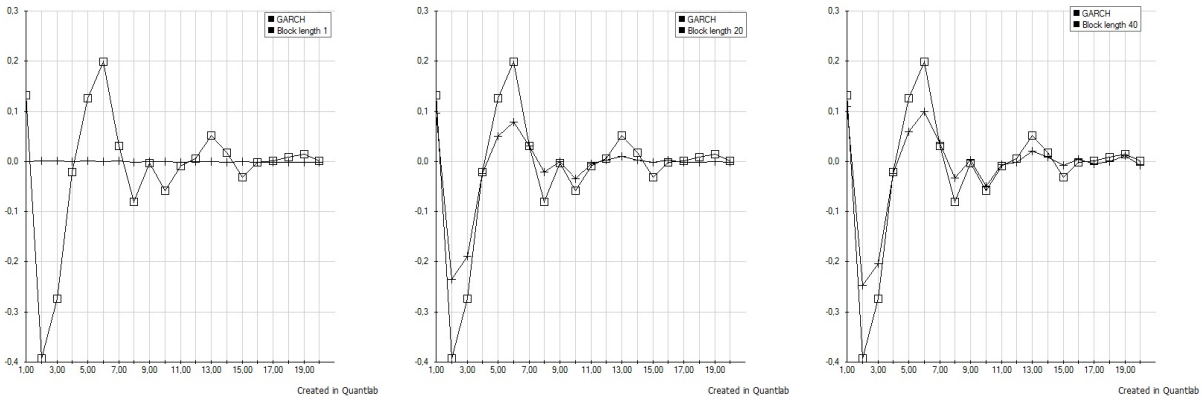
Now construct a stationary bootstrap resample, X^*, by choosing the elements of X with indexes U. To get the m day VaR–estimate with level α, follow the procedures 4–5 from section 5.2. Repeat all of these procedures N times and after that, take the sample mean of these observations. This is the VaR–estimate.

5.4 Implementation of VaR backtest.

To construct a VaR backtest, a large historical data sample is needed. This is in practice a serious problem. When looking at the 99% level VaR for 10–days changes, one is essentially looking at the worst event happening in 1000 days, i.e. roughly 4 years. To get a reasonable amount of data points for this risk measure in a back test many many years worth of data are needed.

There are also a problem with the choice of size of the data used for VaR estimation in the calculations. Not enough data points and the model does not give an accurate estimate and too many data points makes the risk measure to slow in capturing market changes. It is thus necessary to make a compromise.

The construction of a backtest is a simple procedure but takes a considerable time to run. First of, split the data chosen for the backtest into two, not necessarily equal, parts. Think of the first part as the known history and consider the second part as if it had not already happened. The first part of the data is then used in the estimation of VaR, this estimate is then checked against the actual outcome, from the second part of the data. Every time an actual outcome is worse than the VaR estimate, call this an exception, is counted and stored. Last, check if the number of exceptions is significantly different from the α level suggested by the VaR model. If this is the case, the model is not a good measure of VaR, at suggested significance level.



(a) Block length 1 (b) Block length 20 (c) Block length 40

Figure 1: Autocorrelation plots for resampled series (plus) with different block lengths and real data (square).

6 Results

Now how well does these extended bootstrap versions replicate the properties of dependent time series? As seen in section 3.3, the block length is an unknown parameter and thus it might be interesting to see how sensitive the distribution of returns estimates are for different block lengths. In this section, we will study the effects of an increasing block length on the tail of the distribution of varying multi–period returns. We will also perform backtests on time series to get a feel if these estimates are a good estimate of risk. Certain correctional techniques, such as weighting of more recent data and correlation with other time series, will be ignored mainly because of the assumptions that have to be done that risks to "pollute" the data.

6.1 Choice of blocklength

Looking at the autocorrelation plots 1(a),1(b) and 1(c), there is a capture of autocorrelation up to the lag equal to the block length in question. Thus, the VaR–measure seems to only care about correlation up to this point. Further on, looking at the sample variance for multi–period returns, figures 2,3,4, we see a clear deviation from a linear increase in the original sample. Also for the resampled data, the approximation of the variance for different multi–periods for the original series gets better as the block length increases.

As seen, there is a tendency to better and better describe the dependence structure the higher the block length is and intuitively speaking, with a block length equal to the sample size, one would expect to see the same autocorrelation structure as the original sample. The same reasoning goes for the serial variance.

The empirical distribution function of the multi–period returns and the resampled data was compared against each other through the K–S test, see figure 5. There is a clear trend in this data that increasing block lengths up to the number of days in the multi–period returns lowers the K-S distance. As seen there is a local minimum when the block length equals the number of days in the multi–period returns. The distribution fit is also illustrated in the QQ–plots shown in figures

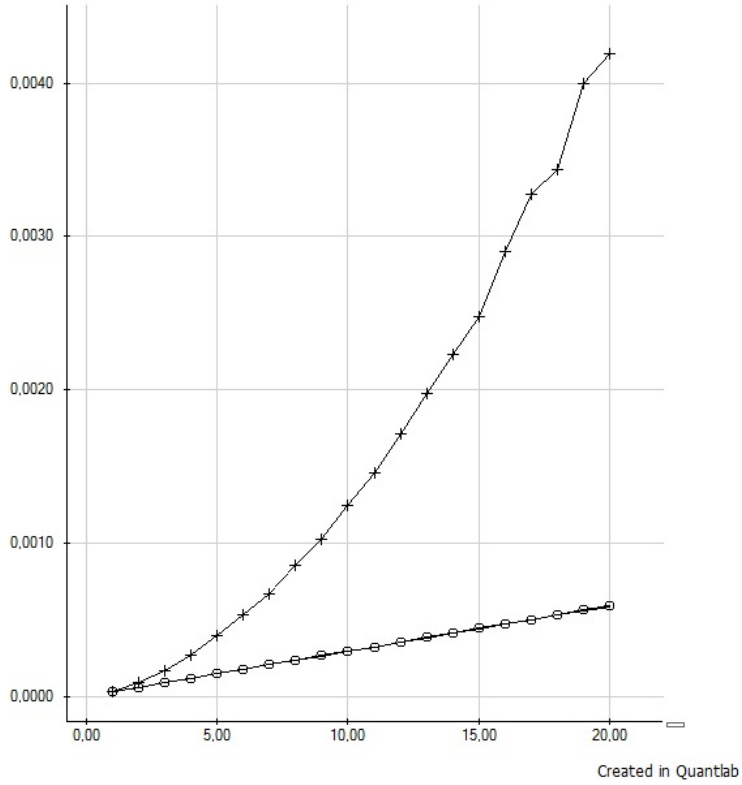


Figure 2: Serial variance for EUR LIBOR 3M fpr block length 1. Plusses are the results for the original series and circles are for the resampled series. The straight line is a linear extension of the 1–day variance.

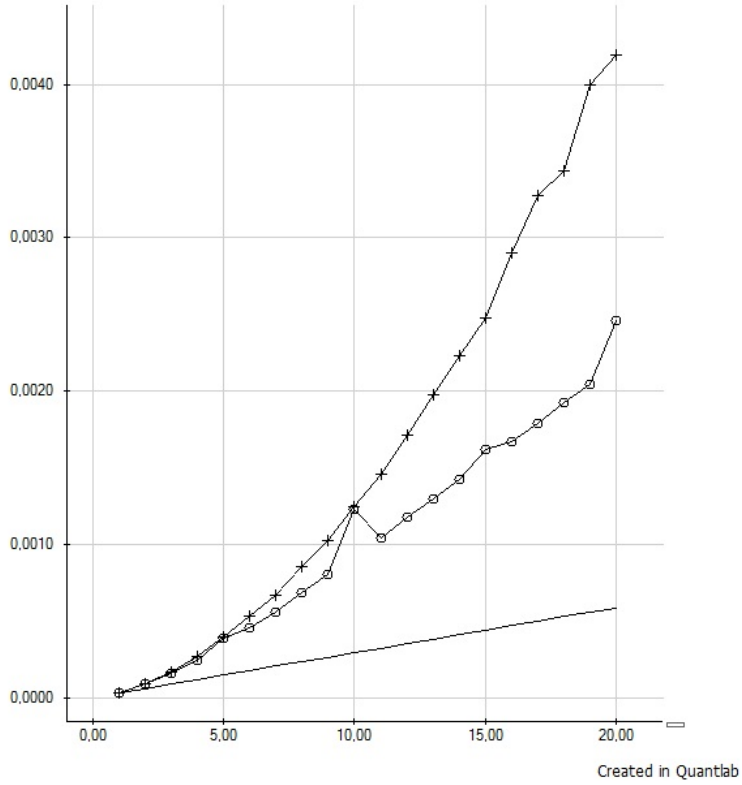


Figure 3: Serial variance for EUR LIBOR 3M for block length 10. Plusses are the results for the original series and circles are for the resampled series. The straight line is a linear extension of the 1–day variance.

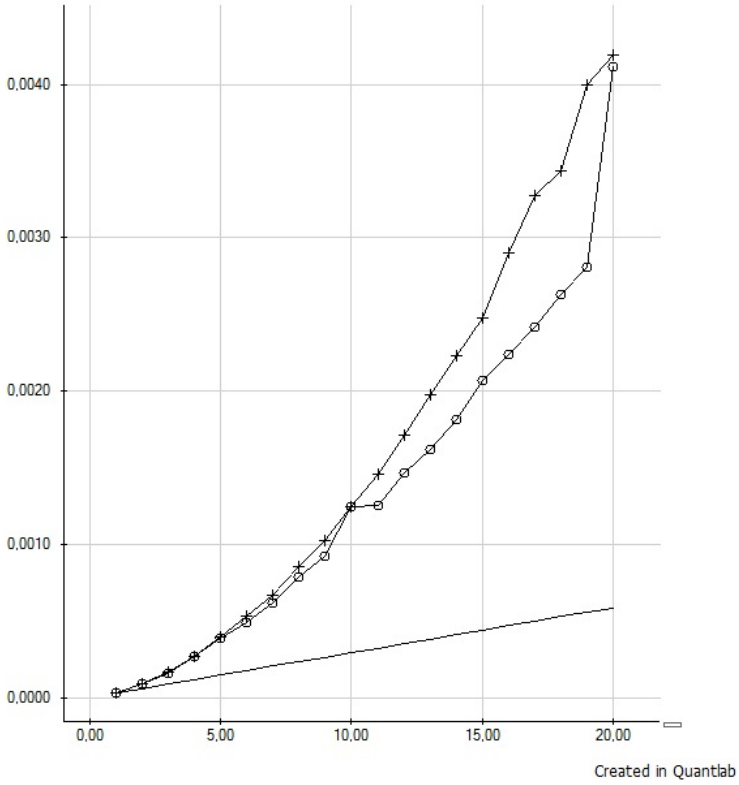


Figure 4: Serial variance for EUR LIBOR 3M for block length 20. Plusses are the results for the original series and circles are for the resampled series. The straight line is a linear extension of the 1–day variance.

8 and 9.

Now shifting attention towards the tails of the distribution function of the multi–period returns, how do they behave as we increase the block length? In tables 5, 6, 7 and 8, the results of this test is presented. Looking at these tables, value–at–risk increases, in magnitude, quickly as the block length increases to the number of multi–period days in question and after this, reaching a plateau. This behaviour could be interpreted as the VaR–method requiring only the correlation structure up until the amount of multi–period days it is looking at.

These results would imply that one should choose a block length high enough to capture enough of the autocorrelation structure and distribution function properties. However, since a large block length decreases the number of drawn blocks, the variance of the estimation would be expected to increase, thus a method for calculating the optimal block length built on MSE minimization were also considered.

The JAB method to estimate the optimal block length for different time series with different multiperiods and different levels of VaR. 100000 resamples were used in this process. The results are presented in tables 2 and 3 and shows similar results as the previous tests. As the multi–period days increases, the suggested block length increases as well at roughly the same size as the multi–period days.

	SEK3M	EUR3M	USD1Y	USD3M	EUR1Y
1	0.79	0.64	0.58	1.81	1.15
5	5.39	7.18	2.97	4.80	3.72
10	9.73	10.00	3.16	8.45	4.05

Table 2: Suggested block length by the JAB–method for different time series for various multi–period returns, $\alpha = 0.95$.

	SEK3M	EUR3M	USD1Y	USD3M	EUR1Y
1	1.44	1.27	1.01	0.90	3.13
5	3.22	6.65	4.61	7.83	4.28
10	6.33	8.49	5.31	12.28	6.00

Table 3: Suggested block length by the JAB–method for different time series for various multi–period returns, $\alpha = 0.99$.

6.2 Backtests

A GARCH$(1, 1)$ process was used to create a large number of data. The parameters used were estimated from the log returns of the 3 month EUR LIBOR note using a maximum likelihood based software. The initial volatility was estimated by the variance of the log returns of the time series. From these parameters, a new dataset was constructed by following the GARCH$(1, 1)$ model and recursively calculating the innovations a_t, as defined in section 2.7. The series produced was used to check the fit of distributions by QQ–plots, as seen in figures 6 and 7.

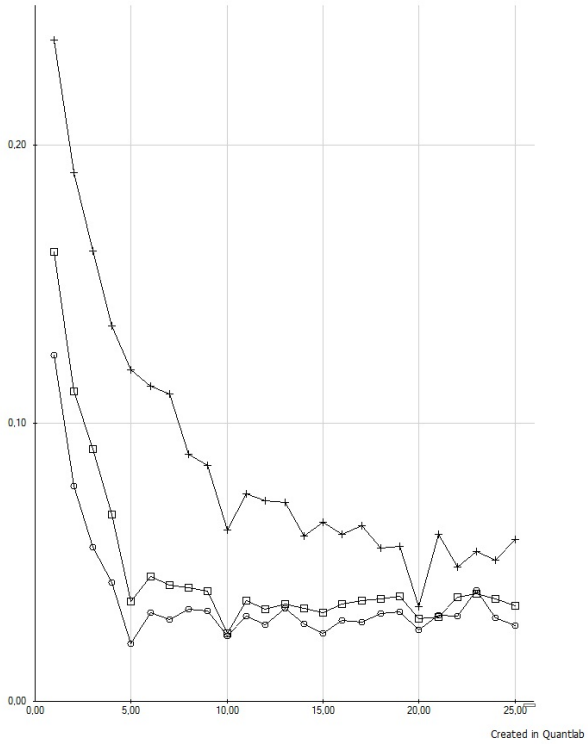


Figure 5: K–S distance plotted against block length for EUR LIBOR 3M. Plus is 20 day returns, square is 10 day returns and circle is 5 day returns.

Further on a 100000 data points long simulated GARCH$(1,1)$ series with parameters $\alpha_0 = 0.00001$, $\alpha_1 = 0.80443$ and $\beta_1 = 0.08256$ was generated for the purpose of backtesting. Each VaR estimate was done by the use of 500 resampled series on a historic data of 1000 data points. As seem in table 4, calculating value–at–risk from this dataset by normal–approximation yields a non–significant measure at the 95% level. However, using the block bootstrap method with block length parameter choosen as twice the size of the number of value–at–risk days currently in question leads to much better results. In fact, the block bootstrap method accepts the null–hypothesis for all value–at–risk levels and time periods. This has to do with the fact that the generated GARCH series is nice to work with, it is stationary and does not show any tendency to "explode".

		0.95	0.96	0.97	0.98	0.99
1–day VaR	Gauss	3550–	3108–	2644–	2138–	1573–
	Bb	4939+	3955+	2990+	1988+	987+
	Sb	4964+	3957+	2979+	1992+	1023+
	$\mathbb{E}$	4950	3960	2970	1980	990
5–day VaR	Gauss	620–	530–	461–	379+	265–
	Bb	939+	728+	547+	370+	174+
	Sb	908+	715+	549+	372+	181+
	$\mathbb{E}$	950	760	570	380	190
10–day VaR	Gauss	276–	228–	199–	161+	119–
	Bb	434+	348+	261+	175+	76+
	Sb	435+	350+	262+	176+	87+
	$\mathbb{E}$	450	360	270	180	90

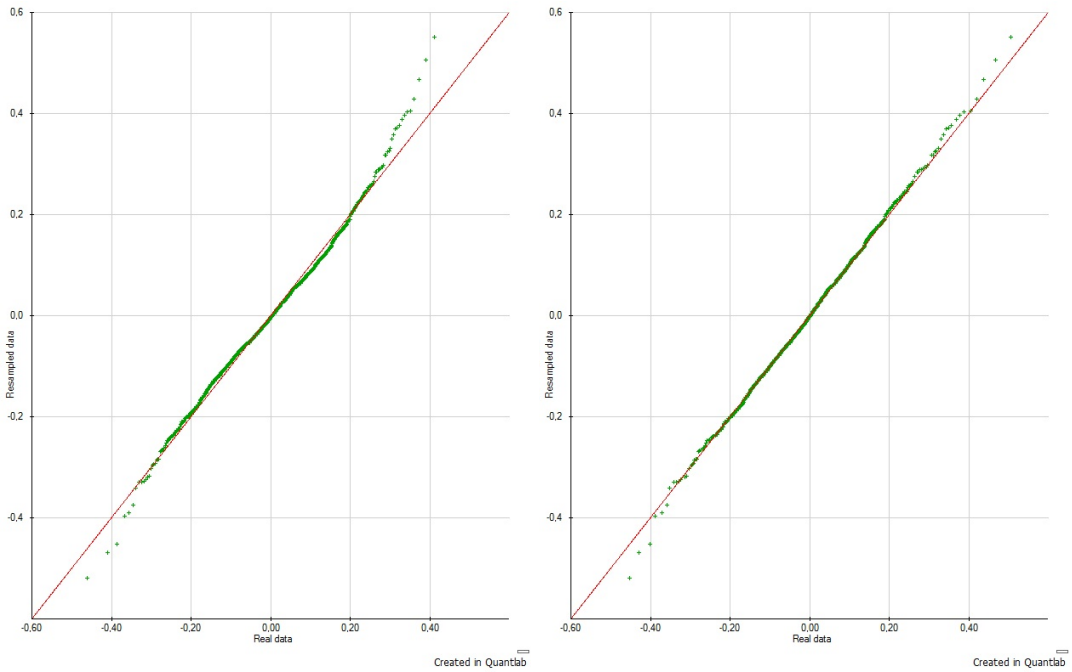
Table 4: Backtest of VaR estimated by stationary bootstrap, Sb, and block bootstrap, Bb, against a gaussian estimate of VaR. A + sign indicates an acceptance of the null–hypothesis in the Kupiec–test. A – sign indicates a rejection. $\mathbb{E}$ is the expected number of exceptions.

	USD_LIBOR_1Y			EUR_LIBOR_3M		
	1	5	10	1	5	10
1	-0.0431	-0.0995	-0.1472	-0.0086	-0.0215	-0.0316
5	-0.0431	-0.1035	-0.1515	-0.0086	-0.0333	-0.0554
10	-0.0432	-0.1046	-0.1605	-0.0087	-0.0358	-0.0626
15	-0.0432	-0.1051	-0.1593	-0.0087	-0.0367	-0.0657
20	-0.0431	-0.1053	-0.1628	-0.0086	-0.0372	-0.0673

Table 5: $\alpha = 0.95$–VaR–calculations for different block lengths for USD LIBOR 1Y and EUR LIBOR 3M by the use of the block bootstrap method.

6.3 Comments on the results

As seen in the Result section, an increasing block length gives a better Kolmogorov–Smirnov fit of the distribution of multi–period returns. The JAB based method, as well as looking directly at the VaR values, however indicates that, block lengths should not be choosen to high. A trade off that



(a) Block length 1 (b) Block length 10

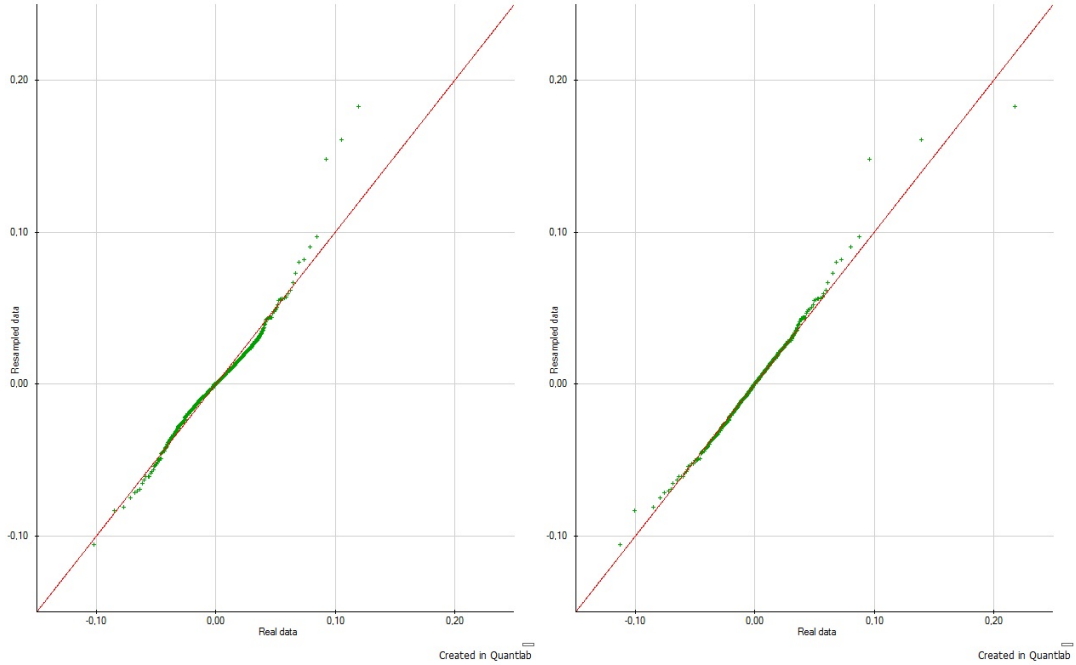
Figure 6: QQ–Plots of resampled log returns against a GARCH$(1,1)$ simulated series with parameters estimated from USD 1Y IR.

	USD_LIBOR_1Y			EUR_LIBOR_3M		
	1	5	10	1	5	10
1	-0.0432	-0.0993	-0.1470	-0.0086	-0.0215	-0.0317
5	-0.0431	-0.1038	-0.1544	-0.0086	-0.0330	-0.0523
10	-0.0431	-0.1046	-0.1586	-0.0086	-0.0358	-0.0671
15	-0.0431	-0.1050	-0.1603	-0.0087	-0.0369	-0.0650
20	-0.0431	-0.1053	-0.1619	-0.0087	-0.0372	-0.0702

Table 6: $\alpha = 0.95$–VaR–calculations for different block lengths for USD LIBOR 1Y and EUR LIBOR 3M by the use of the stationary bootstrap method.

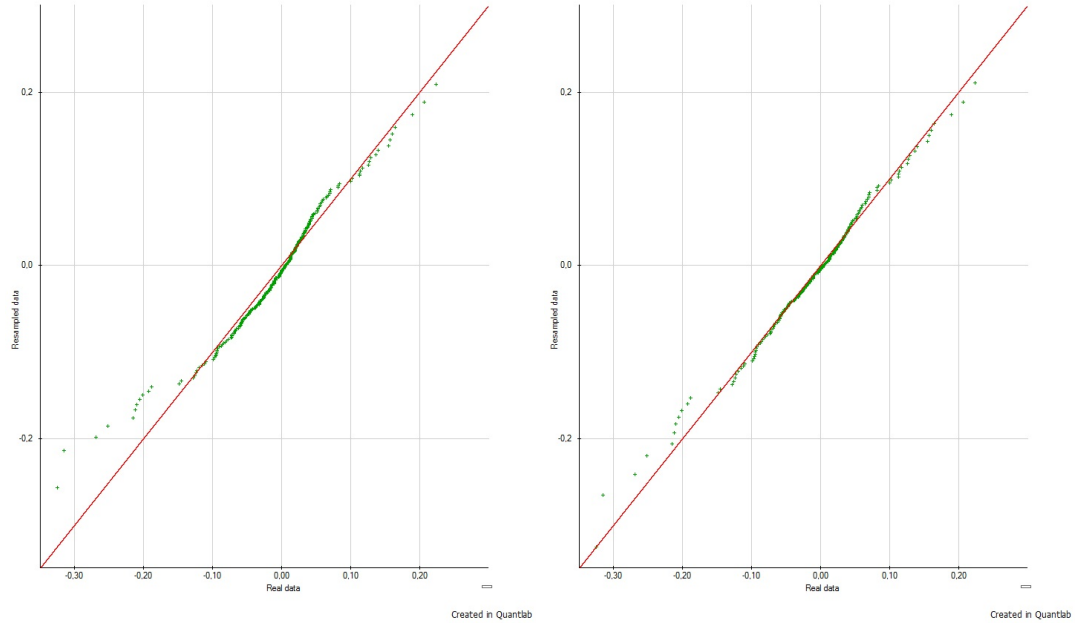
	USD_LIBOR_1Y			EUR_LIBOR_3M		
	1	5	10	1	5	10
1	-0.0756	-0.1509	-0.2125	-0.0168	-0.0365	-0.0546
5	-0.0756	-0.1647	-0.2327	-0.0169	-0.0649	-0.0953
10	-0.0756	-0.1681	-0.2479	-0.0170	-0.0713	-0.1322
15	-0.0756	-0.1687	-0.2473	-0.0170	-0.0732	-0.1288
20	-0.0756	-0.1691	-0.2515	-0.0170	-0.0737	-0.1368

Table 7: $\alpha = 0.99$–VaR–calculations for different block lengths for USD LIBOR 1Y and EUR LIBOR 3M by the use of the block bootstrap method.



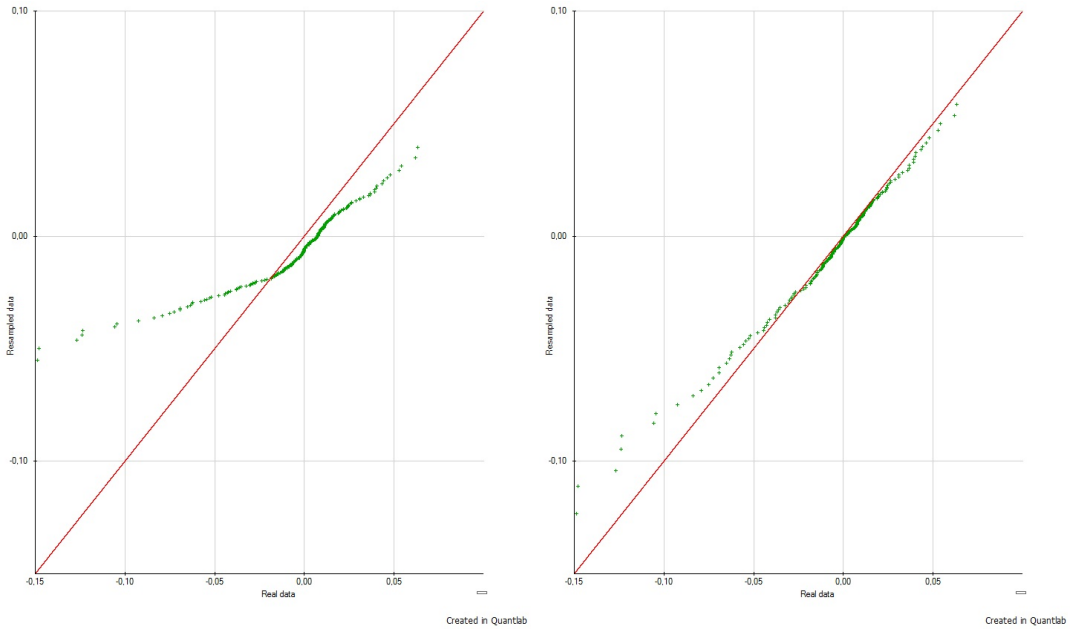
(a) Block length 1 (b) Block length 10

Figure 7: QQ–Plots of resampled log returns against a GARCH$(1,1)$ simulated series with parameters estimated from EUR 3m IR.



(a) Block length 1 (b) Block length 10

Figure 8: QQ–Plots of resampled log returns against the log returns of USD 1Y IR.



(a) Block length 1 (b) Block length 10

Figure 9: QQ–Plots of resampled log returns against the log returns of EUR 3M IR.

	USD_LIBOR_1Y			EUR_LIBOR_3M		
	1	5	10	1	5	10
1	-0.0756	-0.1509	-0.2127	-0.0168	-0.0367	-0.0550
5	-0.0757	-0.1660	-0.2368	-0.0170	-0.0659	-0.1105
10	-0.0756	-0.1681	-0.2451	0.0170	-0.0710	-0.1246
15	-0.0755	-0.1687	-0.2480	-0.0170	-0.0719	-0.1298
20	-0.0755	-0.1689	-0.2503	-0.0169	-0.0728	-0.1314

Table 8: $\alpha = 0.99$–VaR–calculations for different block lengths for USD LIBOR 1Y and EUR LIBOR 3M by the use of the stationary bootstrap method.

seems reasonable is thus a value of block length in the interval of one or two times the multi–period days.

Looking at both the autocorrelation plots and the serial variance plots the autocorrelation structure seems to be preserved in the resamples. Also the serial variance deviation from the independent case is replicated in the same way. The preservation of the "dependence–structure" in the resampled series provides a foundation for the evaluation of dependent time series. As seen not only in the example in the Resampling methods section, but also in the huge difference in VaR calculation for the short term interest rates.

References

[1] Patrik Albin. Lecture notes in the course "stochastic data processing and simulation". Available at `http://www.math.chalmers.se/Stat/Grundutb/CTH/tms150/1011`, 2011–08–17.

[2] Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1:223–236, 2001.

[3] Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[4] LLC Eola Investments. Notes on the bootstrap mehod. Available at `http://eolainvestments.com/Documents/Notes%20on%20The%20Bootstrap%20Method.pdf`, 2011–06–13.

[5] Paul H. Kupiec. Techniques for verifying the accuracy of risk measurement models. *The Journal of Derivatives*, 3(2):73–84, 1995.

[6] S.N. Lahiri. On the jackknife after bootstrap method for dependent data and its consistency properties. *Econometric Theory*, 18:79–98, 2002.

[7] S.N. Lahiri. *Resampling Methods for Dependent Data*. Springer–Verlag New York, Inc, 2010.

[8] Johan A Lybeck and Gustaf Hagerud. *Penningmarknadens Instrument*. Rabén & Sjögren, 1988.

[9] Lionel Martellini, Philippe Priaulet, and Stéphane Priaulet. *Fixed–Income Securities*. John Wiley & Sons, Inc., 2003.

[10] Daniel J. Nordman. A note on the stationary bootstrap's variance. *The Annals of Statistics*, 37(1):359–370, 2009.

[11] Dimitris N. Politis and Joseph P. Romano. The stationary bootstrap. Technical Report 91–03, Purdue University, 1991.

[12] Ruey S. Tsay. *Analysis of Financial Time Series*. John Wiley & Sons, Inc., Hoboken, New Jersey, 3rd edition, 2010.

A C#–code

The programs used in calculations in this thesis are The basic building blocks are presented here.

```
class Norm
{
private WH2006 _myRand = new WH2006();
//Normal-dist r.v. generator using Box-Muller
public double Rnd(double mu, double sigma)
{
double chi = _myRand.NextDouble();
double eta = _myRand.NextDouble();
double z = mu + sigma * Math.Sqrt(-2 * Math.Log(chi)) * Math.Cos(2 * Math.PI * eta);
return z;

}
public static double N(double x)
{
const double b1 = 0.319381530;
const double b2 = -0.356563782;
const double b3 = 1.781477937;
const double b4 = -1.821255978;
const double b5 = 1.330274429;
const double p = 0.2316419;
const double c = 0.39894228;

if (x >= 0.0)
{
double t = 1.0 / (1.0 + p * x);
return (1.0 - c * Math.Exp(-x * x / 2.0) * t *
(t * (t * (t * (t * b5 + b4) + b3) + b2) + b1));
}
else
{
double t = 1.0 / (1.0 - p * x);
return (c * Math.Exp(-x * x / 2.0) * t *
(t * (t * (t * (t * b5 + b4) + b3) + b2) + b1));
}
}
public static double InvN(double alpha)
{

double eps = 0.00000001;
int counter = 0;
double guess = 10;
bool nonconv = true;
if (alpha < 0.5)
{
```

```
guess = -1.0*InvN(1 - alpha);
}
else
{
while(counter<100000 && nonconv )
{
double inv = N(guess);

if (Math.Abs(inv - alpha) > eps)
{
int sign = Math.Sign(alpha-inv);
guess = guess + sign*10.0/Math.Pow(2.0,counter+1);
counter++;
}
else
{
nonconv = false;
}
}
guess = nonconv ? -100000.0 : guess;
}

return guess;
}
}
class DataBase
{

//Static Instance Method
public static List<string> Name()
{
List<string> data = new List<string>();
SqlConnection myConnection = new SqlConnection("user id='';" +
"password='';server=tambora;" +
"Trusted_Connection=yes;" +
"database=arms_yahoo; " +
"connection timeout=30");
try
{
myConnection.Open();
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
try
{
```

```
SqlDataReader myReader = null;
string sql_q =
"SELECT item FROM ahs_data_float WHERE item NOT LIKE '%_OLD' GROUP BY item ";
SqlCommand myCommand = new SqlCommand(sql_q, myConnection);
myReader = myCommand.ExecuteReader();
while (myReader.Read())
{
data.Add(myReader.GetString(0));
}
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
try
{
myConnection.Close();
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
return data;
}
//Instance Method
public List<double> Get(string series)
{
List<double> data = new List<double>();
SqlConnection myConnection = new SqlConnection("user id='';" +
"password='';server=tambora;" +
"Trusted_Connection=yes;" +
"database=arms_yahoo; " +
"connection timeout=30");
try
{
myConnection.Open();
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
try
{
SqlDataReader myReader = null;
string sql_q =
"SELECT val FROM ahs_data_float WHERE item = @Param1  AND"
+"(quote_date BETWEEN '2000-01-01' AND '2011-01-01') ORDER BY quote_date";
```

```
SqlParameter myParam = new SqlParameter("@Param1", SqlDbType.VarChar);
myParam.Value = series;
SqlCommand myCommand = new SqlCommand(sql_q, myConnection);
myCommand.Parameters.Add(myParam);
myReader = myCommand.ExecuteReader();
while (myReader.Read())
{
data.Add(myReader.GetDouble(0));
}
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
try
{
myConnection.Close();
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
return data;
}
public List<double> GetFull(string series)
{
List<double> data = new List<double>();
SqlConnection myConnection = new SqlConnection("user id='';" +
"password='';server=tambora;" +
"Trusted_Connection=yes;" +
"database=arms_yahoo; " +
"connection timeout=30");
try
{
myConnection.Open();
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
try
{
SqlDataReader myReader = null;
// AND (quote_date BETWEEN '2000-01-01' AND '2011-01-01')
string sql_q =
"SELECT val FROM ahs_data_float WHERE item = @Param1  ORDER BY quote_date";
SqlParameter myParam = new SqlParameter("@Param1", SqlDbType.VarChar);
```

```
myParam.Value = series;
SqlCommand myCommand = new SqlCommand(sql_q, myConnection);
myCommand.Parameters.Add(myParam);
myReader = myCommand.ExecuteReader();
while (myReader.Read())
{
data.Add(myReader.GetDouble(0));
}
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
try
{
myConnection.Close();
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
return data;
}
//Destructor
//        ~DataBase()
//        {
// Some resource cleanup routines.
//        }

}
class Resampler
{
//Block length: 1 is 1 data point, 2 is 2 data points and so on..
private List<double> _data = new List<double>();
private int _blockLength = 0;
private int _lData = 0;
WH2006 _myRand;
public Resampler(int blockLength,List<double> data)
{
_blockLength = blockLength;
_data = data;
_lData = _data.Count();
_myRand = new WH2006();
}
public List<double> GetResample()
{
//Init. random nr.
```

```
//SystemCryptoRandomNumberGenerator myRand = new SystemCryptoRandomNumberGenerator();

List<double> resample = new List<double>();

//Draw a starting point in the data.
int m = _myRand.Next(_lData);

for(int i=0;i<_lData;i++)
{
//Generate either a random nr m: 0<=m<=_lData-1 or increase
//m by 1 depending on the block length.
m = ((i+1)%_blockLength != 0) ? m+1:_myRand.Next(_lData);
//Choose the cyclic (m%_ldata) element, 0<=m<=_ldata+_blockLength-1,
//to add to resample.
resample.Add(_data[m%_lData]);
}
return resample;
}
}
class Bb
{
private List<double> _data = new List<double>();
private double _alpha;
private int _blockLength;
private int _varDays;
private int _lData;
public Bb(double alpha,List<double> data,int blockLength,int varDays)
{
_alpha = alpha;
_data = data;
_blockLength = blockLength;
_varDays = varDays;
_lData = _data.Count();
}
private double LinInterpolVaR(List<double> sample)
{
sample.Sort();
int lSample = sample.Count();
double alphaInd = (1 - _alpha) * (lSample - 1);
double lower = Math.Floor(alphaInd);
double x = alphaInd - lower;
double upper = Math.Ceiling(alphaInd);
double uVal = sample[(int)upper];
double lVal = sample[(int)lower];
return lVal + x * (uVal - lVal);
}
private double VaR(List<double> sample)
```

```
{
sample.Sort();
int lSample = sample.Count();
int alphaInd = (int)Math.Floor((1 - _alpha) * (lSample - 1));
return sample[alphaInd];
}
private double Pfe(List<double> sample)
{
sample.Sort();
int lSample = sample.Count();
int alphaInd = (int)Math.Ceiling(_alpha * (lSample-1));
return sample[alphaInd];
}
private List<double> CompData(List<double> sample)
{
int lSample = sample.Count();
List<double> compData = new List<double>();
int nl = lSample/_varDays;
for (int i = 0; i < lSample; i++)
{
if (i >= nl)
{
int k = i%nl;
int j = i/nl;
compData[k] += sample[k*_varDays+j];
}
else
{
compData.Add(sample[i*_varDays]);
}
}
return compData;
}
public double GetVaR(int numSim)
{
double varEst = 0.0;
Resampler myResampler = new Resampler(_blockLength, _data);
for (int i = 0; i < numSim; i++)
{
List<double> resample = myResampler.GetResample();
List<double> compData = CompData(resample);
varEst+=VaR(compData);
}
return varEst*1.0/numSim;
}
public double GetUpperVaR(int numSim, double confInv)
{
```

```
List<double> varEst = new List<double>();
Resampler myResampler = new Resampler(_blockLength, _data);
for (int i = 0; i < numSim; i++)
{
List<double> resample = myResampler.GetResample();
List<double> compData = CompData(resample);
varEst.Add(VaR(compData));
}
varEst.Sort();
int confInd = (int)Math.Ceiling(((numSim - 1) * confInv));
return varEst[confInd];
}
public double GetLowerVaR(int numSim, double confInv)
{
List<double> varEst = new List<double>();
Resampler myResampler = new Resampler(_blockLength, _data);
for (int i = 0; i < numSim; i++)
{
List<double> resample = myResampler.GetResample();
List<double> compData = CompData(resample);
varEst.Add(VaR(compData));
}
varEst.Sort();
int confInd = (int)((numSim - 1) * (1-confInv));
return varEst[confInd];
}
public double GetLinInterpolVaR(int numSim)
{
double varEst = 0.0;
Resampler myResampler = new Resampler(_blockLength, _data);
for (int i = 0; i < numSim; i++)
{
List<double> resample = myResampler.GetResample();
List<double> compData = CompData(resample);
varEst += LinInterpolVaR(compData);
}
return varEst * 1.0 / numSim;
}
public double GetPfe(int numSim)
{
double pfeEst = 0.0;
Resampler myResampler = new Resampler(_blockLength, _data);
for (int i = 0; i < numSim; i++)
{
List<double> resample = myResampler.GetResample();
pfeEst += Pfe(CompData(resample));
}
```

```
return pfeEst * 1.0 / numSim;
}
}
class BackTest
{
private int _varDays;
private int _blockLength;
private double _alpha;
private List<double> _data = new List<double>();
private int _lData;
private int _numSim;
public BackTest(List<double> data,int varDays,int blockLenght,double alpha,int numSim)
{
_numSim = numSim;
_data = data;
_varDays = varDays;
_lData = _data.Count();
_blockLength = blockLenght;
_alpha = alpha;
}
static double Variance(List<double> data)
{
double mean = data.Average();
int size = data.Count();
double var = 0.0;
for (int i = 0; i < size; i++)
{
var += Math.Pow(data[i] - mean, 2.0);
}
return var / (size - 1);
}
private List<double> CompData(List<double> sample)
{
int lSample = sample.Count();
List<double> compData = new List<double>();
int nl = lSample / _varDays;
for (int i = 0; i < lSample; i++)
{
if (i >= nl)
{
int k = i % nl;
int j = i / nl;
compData[k] += sample[k * _varDays + j];
}
else
{
compData.Add(sample[i]);
```

```
}
}
return compData;
}
private static List<double> SubVector(List<double> data,
                                int indStart,int numberOfElements)
{
List<double> subVector = new List<double>();
for (int i = indStart; i < indStart+numberOfElements; i++)
{
subVector.Add(data[i]);
}
return subVector;
}
private double VaR(List<double> sample)
{
sample.Sort();
int lSample = sample.Count();
int alphaInd = (int)Math.Floor((1 - _alpha) * (lSample - 1));
return sample[alphaInd];
}
private double Pfe(List<double> sample)
{
sample.Sort();
int lSample = sample.Count();
int alphaInd = (int)Math.Ceiling(_alpha * (lSample - 1));
return sample[alphaInd];
}
public int NoExceptions(int lHistoric)
{
int lFuture = _lData - lHistoric;
int noChecks = lFuture/_varDays;
int exceptions = 0;
for(int i=0;i<noChecks;i++)
{
List<double> sample = SubVector(_data, i*_varDays, lHistoric);
Bb myBb = new Bb(_alpha, sample, _blockLength, _varDays);
double varEst = myBb.GetVaR(_numSim);
double future = SubVector(_data,lHistoric+i*_varDays,_varDays).Sum();
if(future<varEst)
{
exceptions++;
}
}
return exceptions;
}
public int NoExceptionsPfe(int lHistoric)
```

```
{

int lFuture = _lData - lHistoric;
int noChecks = lFuture / _varDays;
int exceptions = 0;
for (int i = 0; i < noChecks; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
Bb myBb = new Bb(_alpha, sample, _blockLength, _varDays);
double pfeEst = myBb.GetPfe(_numSim);
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future > pfeEst)
{
exceptions++;
}
}
return exceptions;
}
public int NoExceptionsConf(int lHistoric,double confLvl,string method)
{
if (method == "upper")
{
int lFuture = _lData - lHistoric;
int noChecks = lFuture / _varDays;
int exceptions = 0;
for (int i = 0; i < noChecks; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
Bb myBb = new Bb(_alpha, sample, _blockLength, _varDays);
double varEst = myBb.GetUpperVaR(_numSim,confLvl);
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future < varEst)
{
exceptions++;
}
}
return exceptions;
}
else
{
int lFuture = _lData - lHistoric;
int noChecks = lFuture / _varDays;
int exceptions = 0;
for (int i = 0; i < noChecks; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
Bb myBb = new Bb(_alpha, sample, _blockLength, _varDays);
```

```
double varEst = myBb.GetLowerVaR(_numSim,confLvl);
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future < varEst)
{
exceptions++;
}
}
return exceptions;
}
}
public int NoExceptionsPfeConf(int lHistoric, double confLvl, string method)
{

int lFuture = _lData - lHistoric;
int noChecks = lFuture / _varDays;
int exceptions = 0;
for (int i = 0; i < noChecks; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
Bb myBb = new Bb(_alpha, sample, _blockLength, _varDays);
double pfeEst = myBb.GetPfe(_numSim);
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future > pfeEst)
{
exceptions++;
}
}
return exceptions;
}
public int NoExceptionsGaussVaR(int lHistoric)
{
int lFuture = _lData - lHistoric;
int noChecks = lFuture/_varDays;
int exceptions = 0;
double z = Norm.InvN((1-_alpha));
double sqrtVarDay = Math.Sqrt(_varDays);
for (int i = 0; i < noChecks - 1; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
double mu = sample.Average();
double s = Math.Sqrt(Variance(sample));
double var = sqrtVarDay * s * z;
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future < var)
{
exceptions++;
}
```

```
}
return exceptions;
}
public int NoExceptionsGaussPfe(int lHistoric)
{
int lFuture = _lData - lHistoric;
int noChecks = lFuture / _varDays;
int exceptions = 0;
double z = Norm.InvN((_alpha));
double sqrtVarDay = Math.Sqrt(_varDays);
for (int i = 0; i < noChecks - 1; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
double mu = sample.Average();
double s = Math.Sqrt(Variance(sample));
double pfe = sqrtVarDay* s * z;
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future > pfe)
{
exceptions++;
}
}
return exceptions;
}
public int NoExceptionsHistVaR(int lHistoric)
{
int lFuture = _lData - lHistoric;
int exceptions = 0;
int noChecks = lFuture / _varDays;
for (int i = 0; i < noChecks-1; i++)
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
double var = VaR(sample);
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future < var)
{
exceptions++;
}
}
return exceptions;
}
public int NoExceptionsHistPfe(int lHistoric)
{
int lFuture = _lData - lHistoric;
int exceptions = 0;
int noChecks = lFuture / _varDays;
for (int i = 0; i < noChecks - 1; i++)
```

```
{
List<double> sample = SubVector(_data, i * _varDays, lHistoric);
double pfe = Pfe(sample);
double future = SubVector(_data, lHistoric + i * _varDays, _varDays).Sum();
if (future > pfe)
{
exceptions++;
}
}
return exceptions;
}


}
class Blocks
{
private List<double> _data = new List<double>();
private int _lData;
private int _blockLength;
private List<List<double>> _blocks = new List<List<double>>();
public Blocks(List<double> data,int blockLength)
{
_data = data;
_blockLength = blockLength;
_lData = _data.Count();
if (_blockLength < 1)
{
_blockLength = 1;
}
else if (_blockLength > _lData)
{
_blockLength = _lData;
}
}
public void CreateBlocks()
{
// Creates the matrix, _blocks, containing the blocks.
// this.Get(n) gets the n:th block.
for (int i = 0; i < _lData; i++)
{
List<double> blockTemp = new List<double>();
for (int j = 0; j < _blockLength; j++)
{
blockTemp.Add(_data[(i+j)%_lData]);
}
_blocks.Add(blockTemp);
}
```

```
}
public List<double> Get(int n)
{
// Gets the block of length blockLength starting at index n.
// Must run this.CreateBlocks() first.
return _blocks[n];
}
}
class Jab
{
private int _blockLength;
private WH2006 _myRand = new WH2006();
private List<double> _data = new List<double>();
private int _lData;
private int _noBlocks;
private int _numberOfSimulations;
private List<List<int>> _indexes = new List<List<int>>();
private Blocks _myBlock;
private double _alpha;
private int _blockOfBlocks;
private int _varDays;
public Jab(List<double> data, int blockLength,
   int numberOfSimulations,double alpha,int blockOfBlocks,int varDays)
{
_data = data;
_lData = _data.Count();
_blockLength = blockLength > 0 ? blockLength:1;
_noBlocks = _lData / _blockLength;
_numberOfSimulations = numberOfSimulations;
IndexCreator();
_myBlock = new Blocks(_data, _blockLength);
_myBlock.CreateBlocks();
_alpha = alpha;
_blockOfBlocks = blockOfBlocks;
_varDays = varDays;
}
private List<double> CompData(List<double> sample)
{
int lSample = sample.Count();
List<double> compData = new List<double>();
int nl = lSample / _varDays;
for (int i = 0; i < lSample; i++)
{
if (i >= nl)
{
int k = i % nl;
int j = i / nl;
```

```
compData[k] += sample[k * _varDays + j];
}
else
{
compData.Add(sample[i * _varDays]);
}
}
return compData;
}
private void IndexCreator()
{
for (int i = 0; i < _numberOfSimulations; i++)
{
List<int> tempInd = new List<int>();
for (int j = 0; j < _noBlocks; j++)
{
tempInd.Add(_myRand.Next(_lData));
}
_indexes.Add(tempInd);
}
}
private List<double> Resampler(List<int> indexVector)
{
List<double> resampledData = new List<double>();
foreach (int ind in indexVector)
{
resampledData.AddRange(_myBlock.Get(ind));
}
return resampledData;
}
private double VaR(List<double> sample)
{
sample.Sort();
int lSample = sample.Count();
int alphaInd = (int)Math.Floor((1 - _alpha) * (lSample - 1));
return sample[alphaInd];
}
public double MonteCarlo()
{
double mc = 0.0;
for (int i = 0; i < _numberOfSimulations; i++)
{
mc += VaR(CompData(Resampler(_indexes[i])));
}
return mc/_numberOfSimulations;
}
private bool IsEmpty(int blockNr, int start)
```

```
{
List<int> setBlocks = new List<int>();
for (int i = start; i < start + _blockOfBlocks; i++)
{
setBlocks.Add(i);
}
IEnumerable<int> disjoint = _indexes[blockNr].Intersect(setBlocks);
int j = 0;
foreach (int d in disjoint)
{
j++;
}
return j==0?true:false;
}
public double Phi_i(int index)
{
int noSim = 0;
double var = 0.0;
for (int i = 0; i < _numberOfSimulations; i++)
{
if (IsEmpty(i, index))
{
noSim++;
var += VaR(CompData(Resampler(_indexes[i])));
}
}
return var/noSim;
}
public double Mse()
{
int M = _noBlocks - _blockOfBlocks+1;
int N = _lData - _blockLength + 1;
int n = _lData;
double mse = 0.0;
double phi = MonteCarlo();

for (int i = 0; i < M; i++)
{
double phi_i = Phi_i(i);
double temp = (phi *N - (N - _blockOfBlocks) * phi_i) / _blockOfBlocks;
mse += temp*temp;
}
return mse * (double)_blockOfBlocks / (M*(N - _blockOfBlocks));
}
}
```