ROYAL INSTITUTE
OF TECHNOLOGY

Random testing of a market place system

Simulation of a market place

DAVID KARLGREN

Master's Thesis at Division of Mathematical Statistics
Supervisor Cinnober: Noah Höjeberg
Supervisor KTH: Professor Timo Koski
Examiner: Professor Timo Koski

TRITA xxx yyyy-nn

Abstract

A market place system is a very complex software, which makes it very diffucult to test. The functionalities of the system are at wide range and theoretically the number of possible test cases is infinite.

Several methods are developed for the purpose of testing complex system and make sure they meet the requirements of the customers. The most commonly used test strategies are Static testing and Automated testing.

To be able to find low frequency failures and major critical failures, which cause the system to crash, Random testing is agreed to be a very useful tool of testing. For Random testing to be as efficient as possible in detecting errors, the input domain has to be accurate for the system and the output need to be analyzed carefully with an oracle that fits the purpose.

For the input domain to be accurate, an extensive statistical analysis of historical benchmark data is needed in order to validate under which circumstances the software is operating.

The stock exchange market has increased exponentially the last few years regarding the number of orders on the market place. A large reason for this is the increasing number of markets in general and the increasing number of electronical trading platform developed in particular. The ease at which a customer can trade and follow the market flow has evolved enormously. This forces the developers of the software to deliver a solid and robust trading software, that can handle a large variety of orders and market participants at the same time without major critical failures, such as a crash of the system. The consequences of a system crash can be a huge economic disaster as well as a loss in trust, which could lead to decreasing number of customers and in the end to a loss of income.

Referat

Randomtester av handelssystem

Ett börssystem är ett stort och komplext mjukvarusystem, vilket medför svårigheter då systemet ska testas i sin helhet. Systemens funktioner är omfattande, teoretiskt är antalet möjliga kombinationer av scenarion oändliga.

Det finns ett flertal strategier då man vill utföra tester på komplexa system med hänsyn till kravspecifikationen. Vanliga metoder för ändamålet är statiska tester och automatiserade tester.

En strategi för att exponera lågfrekventa buggar och kritiska buggar, vilka kan leda till att systemet kraschar, är Randomtester som visat sig vara ett mycket användbart vertyg. Randomtester är som mest effektiva då en relevant indatadomän är definierad för testerna samt att resultaten av testerna analyseras noggrant, vanligtvis med hjälp av ett orakel.

För att erhålla en tillförlitlig indatadomän krävs omfattande statistiska analyser av historisk ursprungsdata. Analysen belyser under vilka förhållanden mjukvaran är som mest aktiv.

Handeln på aktiemarknaden har ökat exponentiellt de senaste åren, med avseende på antalet lagda ordrar på marknadsplatserna. En avgörande orsak till trenden är att antalet börser har ökat generellt och att de elektroniska börserna har ökat i synnerhet. Att handla på börsen har blivit enklare i takt med att tillgängligheten har ökat. Aktörerna på marknaden kan med enkla medel följa handelsmönster och exekvera ordrar högfrekvent om aktiekursen fluktuerar. Den ökade närvaron på elektroniska börser ställer stora krav på mjukvarutillverkaren av börssystemen. En nödvändighet för att dagens system ska vara konkurrenskraftiga är att de kan hantera ett stort antal ordrar under ett kort tidsintervall på ett snabbt, stabilt och korrekt sätt utan att systemet kraschar. De ekonomiska konsekvenserna av en systemkrasch kan bli mycket omfattande, dels direkt och dels indirekt genom minskat förtroende och färre nytecknade kunder som använder sig av börssystemet.

Contents

Chapter 1

Introduction

1.1 Background

A market place system is a very complex piece of software, which makes it very diffucult to test. The market place systems have a large number of functionalities and theoretically the number of possible test cases is infinite.

Several methods are developed for the purpose of testing complex system and make sure they meet the requirements of the customers. The most commonly used test strategies are Static testing and Automated testing. To be able to find low frequency failures and major critical failures, such as a system crash, Random testing is generally regardered a very useful complement of testing.

For Random testing to be as efficient as possible in detecting errors, the input domain has to be accurate for the system and the output need to be analyzed carefully.

For the input domain to be accurate, an extensive statistical analysis of historical benchmark data is needed in order to validate under which circumstances the software is operating.

The stock exchange market has increased exponentially during the last few years regarding the number of orders and trades. A major reason for this is the increasing number of markets in general and the increasing number of electronical trading platform developed in particular. The ease at which a customer can trade and follow the market flow has evolved enormously. This forces the developers of the software to deliver a solid and robust trading software, that can handle a large variety of orders and market participants at the same time without major critical failures, such as a crash of the system. The consequences of a system crash can be a huge economic disaster as well as a loss in trust, which could lead to decreasing number of customers and in the end a loss of income.

1.2 Thesis description

A large part of developing a software is the procedure of testing and evaluating. Usually both manual tests and automated regression tests are utilized during the development of the software.

The input domain for a market place system is very large and it is impossible to run every single test case. One important test case to develop is a Random test that in a realistic way simulates a trading day to analyze the software's functionality under realistic circumstances.

This master thesis includes a theoretical study of Random testing, an analysis of real trading data in order to get a realistic input domain for the test, implementation of market participants and a model that simulates a trading day. Finally all parts come together in a test phase, where the simulated model is used for Random testing.

At the moment Cinnober Finacial Technology is using a uniform distribution for order volume and a normal distribution for order price to do simulations from. The analysis of the real trading data focuses on whether this is an appropriate model or not, or if it could be replaced by something more accurate.

1.3 Overview

The master thesis is divided in two studies, one theoretical study and one more applied study. The theoretical part is found in chapters two, three, four and five. It contains of general description of a market place and the trading system used in the implementation. It also includes different strategies for software testing in general and for Random testing in particular. The applied study is presented in chapter six and it contains a mathematical analysis of real trading data and a simulation that is used for the implementation of Random testing.

The first chapter gives a short introduction why this master thesis is needed and a description and the goals for the project.

Chapter two describes the basics of an ordinary market place, including market participants, order types and the procedure when placing an order.

The third chapter gives a summary over the trading system that is being used for testing. The inner structure of the software is treated as well as order management.

Chapter four presents an overview of software testing, such as functional and non-

functional testing. Different types of strategies are described more or less in detail, such as Static testing and Black box testing.

Chapter five is solely about Random testing. A detailed description of the input domain, the software under test and the analysis if the output is presented. The oracle strategy for output analysis, the definition of the Poisson process and how to model the input domain are treated. A section about the reliability of Random tests is also presented.

The fifth chapter deals with the implementation of the model. It contains the analysis of the historical values and the modeling of the results of the analysis aswell as the implementation of market participants and Random testing.

Chapter 2

The Market Place

2.1 General

A marketplace is a space, actual or virtual, under the rules of a market. [Wikipedia Marketplace]

A market consist of different systems, institutions etc. where persons and institutions can trade and exchanging goods and services, forming part of the economy. [Wikipedia Market]

2.2 Market Participants

There are several actors on a market place. They all have specific roles to play, like customers, dealers, brokers, market makers and specialist.[Schauer 2006]

Customers

People and institutions who invest or withdraw money when exchanging shares are defined as customers. They are divided into retail customers and institutional customers, where retail customers trade for smaller amounts than institutional customers.[Schauer 2006]

Brokers

A broker is an agent who work for a fee on behalf of their customers to make trading easier for the less experienced participants. Through a broker, buyers and sellers can trade without revealing their interest to each other. The brokers' strategy is to trade risk-free and at the best possible price available. They are giving advice and doing research as well. Brokers are divided into commission brokers and floor brokers, where comission brokers are employees of a member firm and a floor broker works for another member of the firm.[Schauer 2006]

Dealers

A dealer is a risk-taking trader, who work for their own interest. Dealers are usually counterparties to customer traders.[Schauer 2006]

Market makers and specialist

For a given security, Market makers keep a firms bid and ask orders, standing by to exchange the shares at a pre-decided price. Market makers act on behalf of other brokers and dealers or on behalf of their own customers. Their strategy is to sell shares when others want to buy and to buy shares when others want to sell. Market makers give depth and liquidity to the market.[Schauer 2006]

2.3 Orders and order books

An order at a market place is a stated intention from a market participant to buy or sell securities. Several types of orders exists, each with their own characteristics for their specific purpose.[Schauer 2006]

Dependent on the rules at a stock exchange different orders are available.[Schauer 2006] Some usual order attributes combined together add up to:

2.3.1 Orders Without Price Limit

Market Order

A Market order is a buy or sell order without price limitation. The buyer or the seller specifies the quantity of shares to trade. At best obtainable price, according to position (buy or sell), the trade is executed. A Market order has the highest priority in the orderbook, in most cases. The downside is that a Market order can not be cancelled and is executed regardless of how good or bad the price is for the investor at the moment of the scheduled trade.[Schauer 2006]

2.3.2 Price Limit Orders

Limit Order

A limit order is a stated intention to buy or sell shares to a specific price or a more favorable one. Both quantity and price are specified, in some cases even the duration of the order, in terms of time limits (section 2.3.3). Limit orders have low priority and for a trade to take place brokers must fill the order.[Schauer 2006]

Stop Order

A Stop order is also known as a stop-loss order. Stop order becomes a Market order when a pre-decided trigger price is reached or exceeded.[Schauer 2006]

Stop Limit Order

A Stop limit order becomes a limit order when a pre-decided trigger price is reached or exceeded. Both the trigger price for the stop limit order and the price for the limit order must be specified when placing the order.[Schauer 2006]

Peg Order

A Peg order is matched (pegged) against the current market's best bid price, best sell price or the midpoint between best sell price and best buy price. A Peg order is somewhat similar to a Limit order in the sense that it could have a limit price specified. If limit price is reached the order is cancelled.

2.3.3 Time Limit Orders

Time limit orders have a defined duration of time in which they are valid. Some examples are as follows: [Schauer 2006]

Good for day

Good for day orders are valid during the rest of the trading day. They expire when the market place is closed.

Good till cancelled

Good till cancelled order are valid until they are cancelled or if a trade is executed.

Market on opening

Market on opening regards Market orders. They must be traded in the beginning of the trading day.

Market on close

Market on close regards Market orders. They must be traded in a time interval before the market placed is closed.

2.3.4 Combined Orders

Market orders, Limit orders and Time limit orders can be combined in several ways, which also can involve some volume restrictions. These combinations can be made with varying complexity. Some examples of combined orders are: [Schauer 2006]

Fill-or-kill

The whole order must be filled. Otherwise the order is cancelled.

Fill-and-kill

The whole order is filled and then cancelled.

Immediate-or-cancel

The order is traded immediately or cancelled. It can be partitioned and the untraded securities are cancelled.

All-or-none

Similar to Fill-or-kill, except that it is not cancelled if the order is not filled. The order remains in the orderbook.

Iceberg

Also reffered to as a Hidden volume order. A part of the order volume is visible and the rest of the volume is hidden. The visible part of the volume is updated after trade.

2.3.5 Order Matching and Order Books

Order Routing

The procedure from placing an order to the order ends up in the order book is known as order routing. [Schauer 2006]

Order Matching

The sell and buy orders are matched together before a trade takes place. Usually, the price has the highest priority and after that follows time of entry in to the orderbook. [Schauer 2006]

Order Book

The order book facilitates the opportunity for the market participants to watch the development of the market. The order book contains all types of orders. The orders are sorted with respect to priority. When a new order is placed in the order book, order matching is done immediately. If no trade is possible, the new order is ranked in the orderbook queue after priority, which corresponds to the orderbook's order depth. The orders can be visible, which means that other market participants can see the orders, or they can be dark. A dark order is not visible to the public eye and it is most common for very large orders. [Schauer 2006]

Chapter 3

The Trading System

The random tests will be executed on Cinnober Finacial Technologies trading platform TRADExpress. This chapter deals with the parts of the trading system that is of relevance for the implementation of random testing. [Cinnober 2009]

3.1 Order Management

The Order Management component of the trading system records incoming order actions, validates the order, processes it and finally replies back to market participants. [Cinnober 2009]

3.1.1 Order Actions and Order Types

Depending on order type, orderbook and trading state, different strategies are applied. Orders can be placed from one or several order books. Table 3.1 shows some of the order actions that are permitted, including a short description of what type of action it is. [Cinnober 2009]

Order Action	**Description**
Order Insert	Insert a new order.
Update Order	Modify an existing order.
Cancel Order	Cancels an existing order.
Cancel All Orders	Cancels all existing orders.
Suspend Order(s)	Withdraw orders from the orderbook. Orders remains in the Matching Engine.
Activate Order(s)	Activate suspended orders. Time priority equals current time.

Table 3.1. Order Actions available in TRADExpress

There is a large variety of complexity when combining orders. Several attributes are combined when creating the specific order to send in to the trading system. These attributes are listed in table 3.2, along with the options available for each attribute. Some examples of attributes added together gives a Limit order, Market order, Iceberg order, Fill-or-Kill order, Fill-and-Kill order, Stop order and Market on opening order. [Cinnober 2009]

Order Condition	**State**
Transparency	Dark Visible
Price	Limit Market Peg
Price Improvement	Discretionary
Volume	Fill-or-Kill Fill-and-Kill Hidden Volume (Iceberg) Minimum Volume All-or-None
Trigger	Stop Orders
Validity From	Immediate Suspended Good From Time Include in Next Uncross
Validity Till	Various
Combination	Combination orders

Table 3.2. Order Conditions

Trancparency tells whether the order is visible for the public or not. [Cinnober 2009]

3.1.2 Trading State Model

The trading schedule contains a sequence of trading states, each with different attributes. An orderbook can run several trading schedules simultaneously, with varying time intervals and different functions. A simple model is illustrated in table 3.3, showing attributes, trading states and trading schedule in increasing order of abstraction. [Cinnober 2009]

Attributes	⇒	**Trading State**	⇒	**Trading Schedule**
Automatic Matching		Pre Open		Ordinary Market Hours
Processing Sequence		Open		
Allowed Transactions		Post Open		

Table 3.3. Example of a trade state model

3.2 Execution Services

Trading is continously matched in the orderbook. The order is either executed, cancelled or stored, partially or in total. The price is calculated and the priority is matched with existing orders in the orderbook. Price has the highest priority and then follows time of entry. [Cinnober 2009]

3.3 Market Information and Market Management

The information available for the public eye is managed by the matching engine.

3.3.1 Public Order Flow

Market by Order

The order is shown in total.

Market by Level

The price levels are the only information of the order that is being shown.

Transparency

Trading states can have different attributes, such as visible and dark attributes.

3.3.2 Public Trade Flow

Information of public trades. The trades can be anonymous or visible to the public.

Users of the trading system can control different parts of it. If the trading system is running on several markets simultaneously with Order routing, one can turn on/off specific market if any problem occurs or if it is necessary for other purposes. [Cinnober 2009]

3.4 The platform and the architecture

Most of the generic functions exists in the TRADExpress Platform core, which is the center of the running system. The running system contains a Trading Application Multiplexor (TAX), which gather information from the matching engine (ME), data server, query server and some other servers. TAX is the central system in the core architecture. [Cinnober 2009]

3.4.1 Trading Application Multiplexor (TAX)

TAX is a router that redirect ingoing and outgoing messages. The user interact with TAX and TAX interacts with the running systems components, which means that TAX acts as an abstraction between the user and the components. The TAX manages user authorization and authentication. [Cinnober 2009]

3.4.2 Matching Engine (ME)

The core server in the trading system is the ME. It preserves orderbooks and active orders, aswell as perform matching and generating trades. The ME can handle several transactions at the same time. It manages back-up of the system and sending transaction copies to the standby server. If the system crashes, the standby server takes over with very short response time. [Cinnober 2009]

3.4.3 Common Data (CD)

The CD contains product definitions, user database, authorization information, trading schedules etc. It has a up to date backup data server, which takes over with very short response time if there is a system crash. [Cinnober 2009]

3.4.4 Query Server (QS)

The QS preserves a copy of the active orderbooks and orders. It facilitates for ME when large queries arrive. [Cinnober 2009]

3.4.5 History Server (HS)

The HS keeps the information of over night orders and handles queries of historical information. [Cinnober 2009]

3.4.6 Vote Server (VS)

The VS establishes which server that is primary server and standby server, to avoid business integrity issues. [Cinnober 2009]

3.4.7 Daemon

Daemon handles system operations. [Cinnober 2009]

Chapter 4

Software Testing

4.1 General

The purpose of software testing is to improve the quality of the software.[Spillner 2007] Software testing is done to ensure that the software under test meets the requirements and that it works properly. The aim is to find the programs weaknesses and reveal any extreme behaviors that may occur and to correct them.[NASA 2004]

Software testing deals with the question of establishing the (in most cases unknown) level of confidence that is desired for a program in the sense of functionality.[Wooff 2002]

Software reliability is measured as the probability of a failure-free software operation in a specific enviroment in a specific time interval. A failure refers to a behavior of the software that is unsatisfactory with respect to the user's requirements. A bug is a static fault that occurs due to inaccurate code in the software.[Grottke 2001]

The softwares quality is measured with respect to 4.1.1 - 4.1.4: [Spillner 2007]

4.1.1 Functionality

The attributes that measure the qualifications of the software are contained in the concept of functionality. The relation between input and output and specific reactions to some inputs are described as functionality.

Some examples of subcategories to functionality are adequacy, correctness in result and security in terms of unauthorized use of software.

4.1.2 Reliability and Usability

Reliability can be subdivided into maturity, fault tolerance and recoverability. Maturity describes the frequence of failure due to software defects. Fault tolerance is

the ability to recover the software after a failure and stay stable at some desired level of performance.

Recoverability measures the ability for the system to recover a certain level of performance and recover the data lost due to system failure.
Usability describes how the interaction with the software is done in terms of understandability, ease of learning, operability etc.

4.1.3 Efficiency

Efficiency describes how well the software manage to perform its functions. Usually this is measured in use of resources and time to execute the functions of the software.

4.1.4 Maintainability and Portability

Maintainability includes the area of adaptability, ease of installation, adaptation etc.

The absence of faults can never been shown with software testing. If that were possible every single action would have to be tested, which is impossible due to the fact that the input domain is in practice infinite.[Spillner 2007] One can only show the attendence of an error.

4.2 Types of Testing

4.2.1 Functional Testing

Testing that takes the functions of input and output of a system into account is denoted functional testing. The test method when dealing with functional testing is the Black Box model. The functional requirements must be satisfied, which means that the program must do what it is supposed to do. If a test is executed without failures connected to the requirements, the software functionality is considered to be valid. [Spillner 2007]

4.2.2 Non-Functional Testing

Non-functional testing is aiming on validation of the attributes of any given function of the software. The quality of the software is in focus and keywords under nonfunctional testing are reliability, usability and efficiency. The customers satisfaction in using the software are important and the capacity of fast implementation to new hardware or changes in the program. Some examples of characteristics in nonfunctional testing are: [Spillner 2007]

Load test

Load tests validate the software's ability to handle large loads, like handling many users and transactions at the same time.

Performance test

Performance tests measure response time of the system and the ability of fast processing, often connected to incresing loads.

Stability and reliability

Collection of measurements about the mean time to failure or failure rate for a specified user profile. Non-functional testing is usually followed by further functional tetsing.

4.3 Regression testing

Regression testing is done in order to test a software after modifications have been implemented, to ensure that no failures have been introduced as a side effect. The test is executed on already existing test cases. Regression testing can be functional and nonfunctional. It can be executed on all levels of a software, such as component, system and acceptance test.[Spillner 2007]

4.4 Static Testing

Static testing is a manual walkthrough of the parts that is being tested and the result is static analyzed. No test data is used. The aim of static testing is to check for deviations and failures in the specifications. The analysis is done manually. The static test inludes different reviews dependent on what is being examined. A review is a documentation over the test results.[Spillner 2007]

4.4.1 Walkthrough

The aim of a walkthrough is to expose defects, ambiguities and other problems in the software documentation. The result of a walkthrough consists of proposals to software improvements and suggestions of alternative implementations. The program specifications are examined and reviewed. A walktrough is suited for "non-critical" documents. [Spillner 2007]

4.4.2 Inspection

Inspection reminds of the walkthrough, but it is more structured in the approach of the parts examined. The aim is to find vague areas and defects in the specification of the software, to be able to improve the quality.[Spillner 2007]

4.4.3 Technical Review

The technical review focuses on the similarity between the software documentation versus the specifications, the qualifications and the standards.[Spillner 2007]

4.5 Automated testing

Automated tests are a program driven test, doing exactly the same actions each time it is executed. Automation testing covers a very wide range of actions when executed.[Kaner 2003] This can be utilized in regression testing for analyzing a new implementation. The test is known for the old version of the software and by executing the test on the recent updated software one can expose new failures. Manual testing is the opposite to automation testing and the analysis is done by a human being. An important part in automation testing is the use of an oracle. The automation testing have severel advantages to manual testing. Some examples are: [Kaner 2003][Hoffman 2000]

- The test is repeatable.
- The test has faster running time and it is easy to generalize.

  The disadvantages are:

- Automated testa are time consuming to develop.
- Difficult to know what results to expect when analyzing the output.
- Automated tests do the exact same thing each time they are executed.

4.6 Dynamic Testing

Dynamic testing is a computer driven test, which contains of an input data that is analyzed and an executable test program, the object. The test strategy is to choose test cases one by one until it is terminated by a termination criterion, usually when a failure is exposed.[Mayer 2006]

Dynamic testing is done in a systematic way, which means that a well defined test domain is chosen. The procedure of dynamic testing is to define the conditions and the goal for the test, define specific test cases and determine which method that are being used for the test.

The most common ways to approach dynamic testing is either through Black box testing or White box testing. Black box testing regards the software under test as a box where the relation between input and output is being compared and analyzed. How the result is achieved is of minor importance. White box testing on the other hand take the source code in to account when a test is designed and executed. The

importance when analyzing a White box test is to check the internal process and the output.[Spillner 2007] The road that leads towards the results is of interest.

4.6.1 Black Box Testing/Functional

Black box testing is also known as functional testing.

Black box testing is divided in three segments: an input, a black box (unknown internal structure) and an output. From a functional point of view this can be regarded as defining a test case, doing the test and evaluating the results.[Fu 2003]

When designing a Black box test, the specifications are considered when selecting the appropriate test cases. A functional partition of the input domain collects the parts and the partition classes that the software processes in the same way. The input domain consists of all possible values that are eligible as input and those who are ineligible as input. They are partitioned into equivalent classes according to functionality. Each class can then be substituted with a representative number when testing, even the ineligible equivalent classes must be tested. The same procedure can be made for the output values. By defining the preconditions and calculating the expected values, the result can be evaluated and analyzed.[Spillner 2007]

The probability of finding failures depends on the quality of the partitioning and type of test case executed.[Spillner 2007]

Due to the complexity in finding equivalent classes from the specification, the boundary values can be evaluated instead. The boundaries are often critical when testing, due to incorrect interpretations by the software.[Spillner 2007]

If there exist well defined boundaries, evaluating them is a solid way to expose failures.[Spillner 2007]

The degree of test coverage describes the overall software examined. If there exists 20 equivalent classes, then one gets 75% test coverage by examining 15 of them.[Spillner 2007]

State transition testing take historical actions, events and inputs in to account. Historical state diagrams are used in order to design the test. The software under test is executed from its initial state and it changes states depending on which events that are triggered until it hits the end state and the test is evaluated. It is preferable if the test manage to trigger all defined functions in that state at least once.

Partion testing is an abstraction of systematic testing, that divides the input space

into disjoint subdomains. One example of partion testing is path testing.[Hamlet] The purpose of black box testing is to verify that every requirements of the software is fulfilled.[Fu 2003]

4.6.2 White Box Testing/Structural

White box testing is heavily dependent on the internal structure when a test is designed. The source code need to be available. The softwares logic and data flow are examined. The methods of white box testing are Statement coverage, Branch coverage, Path coverage and Test of conditions.[Spillner 2007]

Statement coverage

The coverage level of the test is determined by the quantity of executed statements during a test. To consider a test to be finished the coverage level need to be reached.[Spillner 2007]

Branch Coverage

Instead of focusing on the number of statement executed, branch coverage deals with in which order the statements are executed by the software. Every possible decision of a statement must be examined that it occurs or be determined that it is not covered.[Spillner 2007] For example, this means that both true and false must be evaluated in a boolean statement.[Edvardsson 1999]

Test of Conditions

If a decision of a statement rely on many conditions connected by logical operators, then the test of conditions take the complexity of these decisions into account.[Spillner] This means that both true and false must be executed in an if-statement.[Edvardsson 1999]

Path Coverage

Path coverage is a test to evaluate all possible paths that a software can take when executed, such as repetitions and loops.[Spillner 2007]

Chapter 5

Random Testing

5.1 General

The main goal of random testing, or all testing in that matter, is to hit the failure domain.[Gutjahr 1999] An input is failure-causing if the software under test exposes a failure for this input.[Mayer 2006]

Random generation of test cases does not necessarily have to include software specifications or source code. Random testing is efficient when there is a lack of input data [Hoffman 1998] [Harmen 1995]. The largest benefit of random testing is that a statistical prediction can be made of the significance of a successful test [Hamlet 1994]. It can expose specific failures, which can not be detected by deterministic approaches, due to the randomness in the testing process [Kuo 2007].

The stochastic approach focuses on a probability distribution as input data to the software under test (SUT). The output is treated as a stochastic variable and is analyzed statistically. The analysis is done in comparison with an oracle, where the results from the SUT and the oracle should coincide for the test to be successful [Hoffman 1998] [Harmen 1995].

The uniqueness of random testing is the ability to quantify the significance of a testcase which does not fail, which means that even a successful test is significant [Hamlet 1994].

For a realtime program, as TRADExpress in this case, random testing is not only dependent on the random input and their sequence. A necessarity is also their input spacing and overlap in time, due to the streams of input that is required for a realtime program [Hamlet 1994].

A short description of how Random testing is done as follows:[Hamlet 1994]

1. The input domain is defined. It is usually an infinite input domain, but it can

be restricted to match the software requirements. For example, only define the price of a stock as positive values.

2. The test points are generated pseudo-randomly from the input domain.

3. The test case is executed on the software under test.

4. The output is analyzed. A answer without any failures equals a success. An output including errors equals a failure. To judge whether the result is a failure or a success, a comparison of the result is made towards an oracle. An oracle can have different levels of abstraction.[Gundemark 2005]

5.2 The input domain

A randomly generated input domain characterizes random testing. From a statistical point of view this means that the test can be statistically analyzed and that the test is unbiased in the sense that any systematic error is eliminated due to the absence of human interactions.[Mayer 2006]

The domain is practically of infinite size for a complex system like a market place system. In order to define an appropriate subdomain for the test, the domain is quantified and qualified with respect to the software requirements into different partitions. The pseudo-random input is generated from the subdomains, which can be partioned even further to get an appropriate division of the input domain. A common way to partition the domain is to divide after functionality, which means that each subdomain contains software actions that are more or less the same. Then it is possible to assign a probability to each partition, which is weighted according to how likely each action may occur. This makes the operational profile, where the frequencies in each partition can be seen of as an empirical distribution over the input domain. The operational profile is also called a 'user profile', because it acts like a software user would.[Hamlet 1994, 2006] The operational profile is used to create a model for the test, which describes the expected actions of the software in use. When this model is defined a pseudo-random input is generated. The model in Random testing is a model for the whole input domain.

Hamlet states that the benefits of choosing "unsystematic" input data is: [Hamlet 1994, 2006]

1. "Because there are efficient methods of selecting random points algorithmically, by computing pseudorandom numbers; thus a vast number of tests can be easily defined".

2. "Because statistical independence among test points allows statistical prediction of significance in the observed results".

If a measurement is repeated over many trials the systematic variations are "averaged out" and the systematic error is removed. When systematization is given up, the significance of testing is measurable.[Hamlet 1994] If an operational profile is available, the test is pretty straight forward. On the other hand if the inner structure and the functionality of the software is totally unknown, one can use the uniform distribution to generate the pseudo-random input. Hamlet states that because the domain is infinite and the random input only is a very small part of it, the use of the uniform distribution can be adequate as a modell. The test can be a success in pointing out failures, but in worst case a mismatch with no significance at all.[Hamlet 2006]

For a interactive, realtime program like a market place system the input must be chosen so it contains a sequence of events. The time between events and the different actions that follow after each other must be evaluated aswell. The time between the events can be simulated through a pseudo-random input from an uniform distribution or any other appropriate distribution.[Hamlet 1994, 2006] The conclusion is that a random test with an existing operational profile may test the software's functionality in a more adequate way. But if no operational profile is available, the use of any other distribution at hand is more efficient in exposing failures than if the random test is not used, due to the lack of knowledge of the software.

The operational profile in this master thesis is simulated from a model that is based on real historical trading data, collected from a similar software in use.

5.3 Software under test (SUT)

5.3.1 Modeling the Input

When creating an input for the test, historical data is analyzed. The aim of the analysis is to gain distribution functions and time dependent processes of the relevant parameters, such as order volume and order price for the actors on a marketplace. The input is generated from simulated model. For a realistic model, one should even include some time variant parameter when simulating a trading day, such as different loads on the trading system during a trading day. When the simulation is executed, the underlying time variant parameter decides the frequency of the trades, randomly or deterministic.

5.3.2 Modeling actors

The input domain for a market place system is very large, due to a large variety of order combinations for each actor. The actor on a marketplace is known as a market participant with a specific user profile. The user profile contains around 20-30 probabilistic actions, each with 2-3 randomized parameters such as order volume, order size, distance from spread etc. A simulation scheme for order insertion is used

for the actors. They execute their actions independently and they are able to trade with each other. This is possible because the actors only regarding their own profile and the current market condition.[Sellberg 2003]

When preparing a simulation different actors and probabilities are pre-defined.[Sellberg 2003] The pre-defined probabilities is determined by the analysis of the historical data.

Random simulations requires post-analysis of the output. Trading simuations are usually very complex due to the interactions between the actors and the different combinations of order types, order prices etc., which makes it difficult to evaluate. Sellberg is listing three types of evaluation that are commonly used: [Sellberg 2003]

1. Trivial tests, whwich are easy to evaluate. An example of a trivial test could be to do a test case with negative values for order volume or order price.

2. The actors can contain simple explicit tests, such as evaluating if a trade takes place after an order is sent into the system.

3. Major failures, like system crash.

5.4 The Oracle

The oracle's role in random testing is to generate the expected values for the pre-defined software actions. This is used to evaluate the test results from the software under test (SUT). Some characteristecs for an oracle in order to evaluate a test case are according to Hoffman: [Hoffman 2001]

Completeness of the information

The completeness of the information covers all the information of a test case that involves the input, the output, a successful test and a test failure. Completeness can be at a wide range, from no prediction to a complete prediction of the software under test (SUT).[Höjeberg 2007]

Completeness is a measure whether the information is sufficient and what type of error that is expected.

Accuracy of information gathered from a test

The oracle's correspondence to SUT, both arithmetic accuracy and statistically similarity.

Independence between the oracle and the SUT in the meaning of algorithm, system platforms and operating enviroment etc.

More resemblance between the oracle and SUT implies a more complex oracle, which causes more errors.

Usability of the oracle and/or its results

The usability of the oracle describes what type of information the oracle is generating and the location of the exposed failures.

Maintainability of the oracle

Maintainability covers the modification possibilities by the oracle when changes in SUT are implemented. Less complex oracle means that it is easier to modify, which is also less expensive to develop and has less internal errors.

Complexity

Complexity describes the interaction between the oracle and SUT and how much of the functions and the domains that are covered during the test case.

Temporal relationships

Executable time of the oracle in means of generating results and comparing those results to that of the test case. It is correlated with the abstraction of the oracle.

Costs

The oracle's total costs in means of development, maintenance and analysis.

The more complex an oracle is, the more accurate and complete it is, which means that it provides better expected results. The downside is that the more complex the oracle is, the more errors will be traced to the oracle and not to SUT. This means that the discrepancy in the result between SUT and the oracle is due to the oracle's internal structure. [Hoffman 2001]

An oracle can have different abstractions dependent on the requirements of the test. The varity of the oracle streches from the strategy of no oracle in one end to a true oracle in the other end. Some of the oracles that Hoffman discusses are: no oracle, true oracle, consistent oracle, self-referential oracle (SVD) and heuristic oracle. [Hoffman 2001]

5.4.1 No oracle

The advantages of a no oracle strategy are a fast test run and that it is easy to implement. The costs are minimized in the aspect of development and maintaining the oracle. The disadvantages are that the results from the SUT never can be

compared to an oracle, which implies that only spectacular errors can be detected, such as a system crash. The correctness of the result is never verified. Hamlet states that useful testing must rely on an effective oracle. That is not the case for the strategy of no oracle. Though, it can be a way of testing the softwares functionality and reliability. [Hoffman 2001] [Hamlet 1994]

5.4.2 True oracle

The true oracle is an independent implementation of the SUT, using different algorithms, platform etc. It is given the same input as the software under test. The true oracle verifies the results from the SUT by comparison between the test results and the oracle's reproduced expected results of the test. The correctness of the results gain more confidence the less the true oracle have in common with the SUT.

A small input is preferred to be used when running a test because true oracles can be slow. This can be done by generating the random input from a domain where the true oracle is solid and accurate. True oracles are expensive to develop and to maintain. [Hoffman 2001]

5.4.3 Consistent oracle

A consistent oracle is based on previous test results when it is defined. The strategy of consistency is well used in automated regression tests, particularly when evaluating the changes from an old version of the software to a more recent one. The evaluated result should expose the changes and differences between the oracle and the SUT.

A consistent oracle does not state whether the result is correct or not. The advatage of this approach is that the correctness is irrelevant, which means that enormous quantity of pseudorandom data can be generated and compared. The changes and differences will be exposed, but old undetected error may go unnoticed. Consistent oracles can be an equivalent product, a software from a different platform or an older version of the software under test. [Hoffman 2001]

5.4.4 Self-Referential oracle(SVD)

For a self-referential strategy the input data contains the expected output. The output is verified through data matching. Repeatable testing is done by adding a "seed" in generated pseudo-random input. The advantage with the "seeding" is to create a record of input populations with specific characteristics, the characteristics are embedded in the record itself. Independent analysis reveals problems and inconsistencies.[Hoffman 2001] [Höjeberg 2007] [Gundemark 2005]

5.4.5 Heuristic oracle

A heuristic oracle picks up the results likelihood of being incorrect or correct. The strategy aims on verifying the major characteristics in the test domain. For the major characteristics exact test results are produced. The results can be verified by consistency checks and simple algorithms based on a heuristic approach. A heuristic oracle is quick and simple to implement. It can reveal many classes of errors in short time, especially efficient in finding boundry conditions and special values failures. The downside is that correct results can be flagged as errors and incorrect results can be accepted. A heuristic oracle can not reveal all type of errors. A heuristic oracle can be built into the software under test.[Hoffman 1998, 1999, 2001]

5.5 Reliability and theory

5.5.1 Reliability and prediction

Software reliability is defined as "the probability of failure-free operation of a computer program for a specified time in a specified environment". In practice, the modelling is based on establishing the probability of failures in that specified interval of time, and what the expected time to and between those failures are.[NASA 2004] There are two types of reliability models:[NASA 2004]

Prediction models

The model predicts the reliability of the software under development. The goal is to predict a fully developed software's reliability. One can start this reliability modelling as soon as the requirements of the software are determined.

Estimation models

The approach is to evaluating the most recent software in operation, usually done under the test phase. The aim is to estimate the quantity of remaining failures and the time to and between failures.

The major advantage of random testing is that statistical prediction can be made of the significance of a successful test. For a realtime program this means, according to Hamlet that a valid statement after random testing could be: "It is 99% certain that the program, PR, will have a meantime to failure that is greater than 10000 hours of continuos operation", or equivalently $\mathrm{P}(PR > 10000) > 0.99$.

Meantime to failure, MTTF, is the expected time to failure of a system, when not regarding the time it takes to repair it.

Again, regard a program, PR, with a (constant) failure rate, θ, denoted as the percentage of the inputs that causes failures within the input domain.[Mayer 2006]

MTTF can be derived as:

Define θ as:

$$\theta = \frac{f}{d}$$

where f is the number of inputs in the domain d that causes failures. d is the size of the input domain.[Gutjahr 1999]

θ denotes the probability of a failure for one test point, which gives the probability (1-θ) that the test is a success.

For k number of tests, from the same input domain, the probability that a test is successful is:

$$P_{success} = (1-\theta)^k \tag{5.1}$$

This is equivalent to say that in $\frac{1}{\theta}$ test runs only one failure will be expected. $\frac{1}{\theta}$ is referred to as the confidence probability.

The probability of the detecting at least one failure with k test points is then: [Hamlet 1994]

$$P_{failure} = 1-P_{success} = 1-(1-\theta)^k = 1-(\frac{|D^{fail}|}{|D|})^k = 1-(1-\frac{f}{d})^k = 1-(1-\frac{\sum_{i=1}^{k} f_i}{\sum_{i=1}^{k} d_i})^k,$$

where $\sum_{i=1}^{k} f_i$ is the sum over k subdomains in f, according to the partition of f. The sum of f_i over k subdomains is equivalent to f. The same statement is valid for the failure-causing domain d.

Derivation of 5.1 gives:

$$P_{success} = (1-\theta)^k \Rightarrow k = \frac{log(1-P_{success})}{log(1-\theta)} \tag{5.2}$$

Solving 5.1 for $\frac{1}{\theta}$ yields:

$$\frac{1}{\theta} >= \frac{1}{1-(P_{success})^{\frac{1}{k}}}, \tag{5.3}$$

which is equivalent to Mean time to failure (MTTF), because $\frac{1}{\theta}$ is the expected value.

k can be intepreted as the quantity of the input domain that is required in theory to gain a confidence level of $P_{success}$.[Gutjahr 1999, Hamlet 1994] This is valid for a non-realtime program and realtime programs if a more complex operational profile is available.[Hamlet 1994] For real-time programs there is a sequence of input data arriving randomly.

Suppose that the number of occurrences of an event E in the time interval (0,t] are independent. In this case the event E is whether a customer place an order or not. Each single occurrence is Bernoulli distributed, either the event E occurrs or it does not, with some probability p. If an event occurrs it is a success and otherwise it is a failure. The Binomial distribution is the number of successes in independent repetitions of Bernoulli distributed trials. If the time interval is divided in n disjoint subintervals like, $(0, \frac{t}{n}], (\frac{t}{n}, 2\frac{t}{n}], ..., (\frac{t}{n}t, t]$, where n is very large. The probability that an order is placed in such time interval is then very small. The events are independent in the disjoint time intervals and the number of time intervals is very large. Then it is possible to approximate the number of occurences of inserted orders in the time interval (0,t] as being Poisson distributed. The Poisson approximation is valid for at most one occurence of event E (an order is inserted) in each small time interval. The Poisson process is a discrete stochastic process in continous time, $\{X(t), t \geq 0\}$, and is defined as:[Gut 1995]

$$X(t) = \text{ \# occurrencies in (0,}t\text{]}. \tag{5.4}$$

5.5.2 Definition and properties of the Poisson Process

- The Poisson process is denoted as,

$$X(t) \in Po(\lambda t), t \geq 0, \tag{5.5}$$

  where λ is the intensity of the process.

- X(0)=0 and there exists a $\lambda > 0$ such that

$$X(t) - X(s) \in Po(\lambda(t-s)), \text{for } 0 \leq s < t. \tag{5.6}$$

  At time zero the process equals zero and the equation implies that the increments are independent for some $\lambda > 0$.

- The time to the k:th occurence τ_k, for $k \geq 1$, are independent Exp($\frac{1}{\lambda}$)-distributed random variable.

- The probability function of a Poisson process is:

$$P(X(t) = k) = e^{-\lambda t}\frac{(\lambda t)^k}{k!}, \text{for } k = 0, 1, 2, ... \tag{5.7}$$

5.6 Discussion of random testing versus systematic testing/partition testing

There are especially two cases where random testing is a viable choice: [Hamlet 2006]

Sparse sampling

When the input domain is large and unstructured, which means that it is pointless to partition it into subdomains. This statement is theoretically established due to many studies in the field.

Persistent state

The majority of softwares have a storage of information that is preserved between each test case. This storage is utilized when testing by forcing the software under test to enter that persistent state and perform the desired tests from an initial condition. In general, the test contains a data sequence, which makes the software go through different states whlie the test is active. The final state terminates the test, where the result is gathered. The input sequence is preferably random, due to some advantages in analyzing the test results, which makes Random testing viable. [Hamlet 2006]

Systematic testing is more expected to expose failure than random testing in general. The conclusion for this statement is that systematic testing aiming on a pre-judgement of a high risk section where failures can arise. The priority of such a test is to trigger a "pre-defined" failure on the selected partition. Random testing on the other hand does not make such assumption, which intuitively leads to the conclusion that systematic testing expose more failures than random testing. But in practical use of the methods stated it is not that big difference between the methods, even though the assumptions are favorable to systematic testing.[Hamlet 1994] [Gutjahr 1999] For systematic testing one must know before the test case is permuted what type of failure one want to expose. Random testing on the other hand is more suited to expose failures that are complex and unique for the system under test.

According to Hamlet partition testing is more efficient to expose failures than random testing in a general case. The difference between the methods can be decreased through selecting more test points in random testing and the difference can be increased by choosing a partion that is known to have a high failure rate in the selected domain. The conclusion is that if the system has subdomains with a high failure rate, then partion testing is a good choice in the sense that it concentrates the failures within the chosen subdomain. [Hamlet 1994]

5.6. DISCUSSION OF RANDOM TESTING VERSUS SYSTEMATIC TESTING/PARTITION TESTING

Random testing is the favorable choice over partion testing if one makes the assumption that the failure rates are deterministic. This conclusion is even stronger if the partition includes few large or many small subdomains.[Gutjahr 1999]

Hamlet states that for successful use of random testing the following conditions are neccessary:

- The test is at system level.
- There exists an operational profile.
- Representative pseudo-random input domain.
- Use of an effective oracle.
- "The worst misuse of random testing occurs when the method is not used at all".

Chapter 6

Implementation and Results

6.1 Introduction

This chapter contains the implementation of a market participant. The content in the development of the actor is an analyze of historical trading data, which is used for modeling a stock market, in which the traders places their orders. The purpose is to get a realistic input domain for random testing of the software.

The simulation is done as a part of Random testing. The test model is sent into the trading system, which should be able to simulate a realistic trading day, where the actors trade with each other. A short description of the process is presented in section 6.1. A more detailed description of each moment of the development is presented in separate sections. The theory in this section is found in [Ljung 1987] and [Brockwell 2002].

6.1.1 Historical values

Log-files over past transactions are filtered and sorted with respect to orderbooks and type of bid order. The type of bid orders are buy and sell orders. The result is presented in three columns: Order price, order volume and time for order.

6.1.2 Modeling and simulation

From the sorted log-files, several mathematical models are estimated and analyzed. The aim is to find a suitable model that simulates an actor well, when it comes to forecast the parameter of the next trade. The goal is to find any appropriate model that can be utilized for pseudo-random simulations of the input domain.

For a time dependent model, the autoregressive time series of order one is analyzed. The restriction to order one is due to the test framework's lack of memory. Only the most recent trade is saved in the test framework.

The analysis of the residuals is concentrated to determine whether or not the residuals can be regarded as uncorrelated and what type of distribution that can be applied to them.

Another way of analyzing the benchmark data is to assume that the data originates from pure noise, which implies that the sample is independent identically distributed noise. The analysis in that case focusing on finding an appropriate probability distribution function, from which pseudo-random numbers can be generated.

At the moment Cinnober FT is using a normal distribution when generating order price and an uniform distribution when generating the order volume for random testing. Price and volume are assumed to be independent. The frequency of orders sent into the system is assumed to be Poisson distributed.

6.1.3 The market participants

Under test, several actors are logged into the market place system under test with a predefined number of actions before logging out. The actions are in general different types of orders, where the actor can insert a new bid or ask order, change an excisting order or cancel one or all orders. When a new order is inserted or changed, the order volume and the order price is randomly generated from the probability models determined from the analysis of the benchmark data. The frequency of order insertion is generated from a Poisson distribution. The market participants trades with each other independently. They listen to the market flow and put orders according to the market price.

6.1.4 Implementation of market participants and random testing

The implementation is made with close attachment to reality. The mathematical analysis of the historical trade data is used to imitate a real trading day as well as possible. The market participant that are implemented is a trader, that runs as a thread in Java. This makes it possible to create several actors, that can trade independent of each other. The Random test program is run towards Cinnober finacial technology's trading system TRADExpress.

6.2 Historical values

To get a realistic model, real trade data is needed to be analyzed. The trading data is collected from the trading platform Turquoise, during a period from 9th of February to 19th of February 2009. The log-files contains a lot of information that is unnecessary and it is submitted in an unfriendly way with respect to analysis of the data. Therefor a major filtration is required. Due to the characteristics of an actor, the most important values to filter are the order price, the order volume and the time for inserting the order, recalculated to seconds. Time t=0 corresponds to

trade-day-opening, 09:00:00 CET.

The filtering was made in Java in three steps. The first program makes a rough filtration, sorting out unnecessary data and data sensitive for integrity reasons. The second filtration is aiming on the keyword INSERT, which means that only orders of the type INSERT is passing through. The type INSERT means that a market participant is placing a new order or changing an existing one. The third and final program filtrates each order with respect to orderbook. It also creates three diifer-ent textfiles, one with bid orders, one with ask orders and one where bid and ask order are combined.

Each orderbook contains a specific type of instrument, so it is possible after the filtration to view the fluctuations of a specific instrument. The bid and ask order filtration is made to illuminate the differencies between the bid and the ask order, if there are any. The final result is presented in a three column textfile for each orderbook, for example: XXXXBID.txt, XXXXASK.txt, XXXX.txt etc.

6.3 Fitting a model to order data and simulation

The main purpose for this section is to estimate appropriate models for order price and order volume at some time t, when the market participant is at time (t-1). This can be seen of as a one step prediction of a certain instrument, where the prediction is equivalent to placing an order.

For time variant assumption, AR(1)-processes are analyzed for both order price and order volume. As stated above, the analysis is concentrated to determine if the residuals are uncorrelated and independent. If it can be stated that the residuals are uncorrelated, a probability distribution function is fitted to the residuals. Sev-eral tests to validate the probability distribution function is executed in order to determine it's correctness.

For the assumption of independent identically distributed noise, IID-noise, a prob-ability distribution function is fitted to the benchmark data. As for the residual analysis, several tests are done to determine the appropriateness of the model.

The most important aspects of the model are simplicity when implementing and accuracy in the predictions as far as it is possible. The trading data is collected from stock exchange trading, which makes the model appropriate for stock exchange simulations.

6.3.1 Preparing the benchmark data for analysis

Due to the large amount of data gathered under each trading day, it is necessary to screen off the test set of data. This was done by choosing one orderbook to

estimate a model from and another orderbook to validate the estimated model with. For time series, trend and/or seasonality of order volume and order price were eliminated using box-cox transformation, with λ=0, equation 6.1[Brockwell 2002]. The arithmetic mean was subtracted from the data in order to normalize the time series around zero.

$$f_\lambda(U_t) = \begin{cases} \lambda^{-1}(U_t^\lambda - 1), & U_t >= 0, \lambda > 0, \\ \ln U_t, & U_t > 0, \lambda = 0 \end{cases} \tag{6.1}$$

The data that are being used for estimation of a model are displayed in figure 6.1 and figure 6.2. The data that are being used for validation of the estimated model are illustarated in figure 6.3 and figure 6.4.

By using the System Identification Toolbox in MATLAB it is possible to fit a model to a data set, by choosing the corresponding data set for estimation and validation. The autoregressive processes are applied on order volume and order price. The Yule-Walker estimation of an autoregressive process is also applied to the benchmark data, section 6.3.2. The Yule-Walker estimation is mainly used for reason of simplicity. It is a simple way to get an opinion about the process, according to the order of magnitude one could expect when analyzing the time-series.

The theory of the time series models is treated in section 6.3.2. The analysis of the estimated models focuses on prediction efficiency and residual analysis.

6.3.2 Notation and theory

This section lines up the major definitions and notations used in the analysis of the benchmark data, such as the definition of the autoregressive process and so forth.

The notation for the parameters in the autoregressive process of order p is:

- $X(t)$ equals the output argument at time t.
- $\phi(B)$ equals the backward shift operator B, operationg on $X(t)$. The estimated constants is included in vector ϕ.
- $Z(t)$ is the white noise at time t, which has some distribution with mean 0 and variance σ^2 under the assuption that the estimated model is correct.
- $\theta(B)$ equals the backward shift operator B, operationg on $Z(t)$. The estimated constants is included in vector θ. This is only used for autoregressive moving average processes, ARMA-process. The autoregressive process is an ARMA-process, with $\theta = 1$.

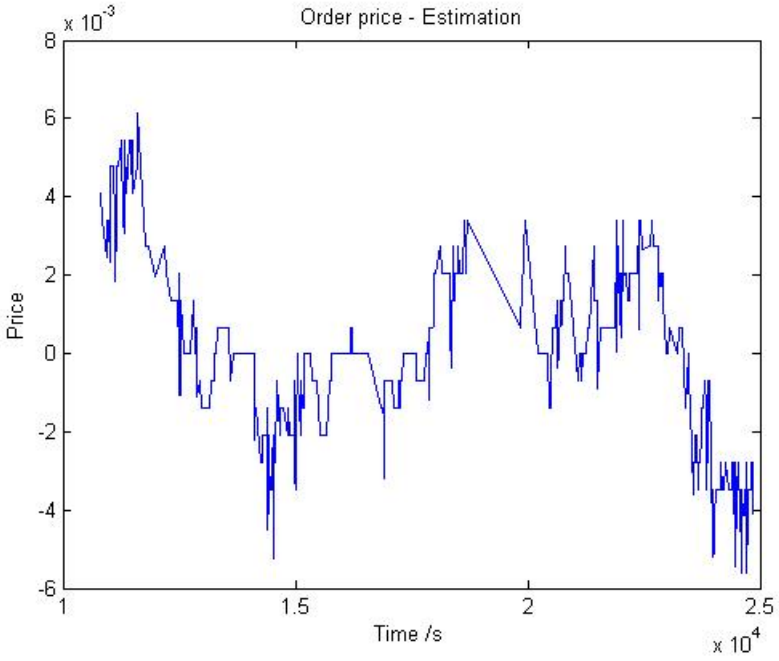

Figure 6.1. Order price - Estimation data

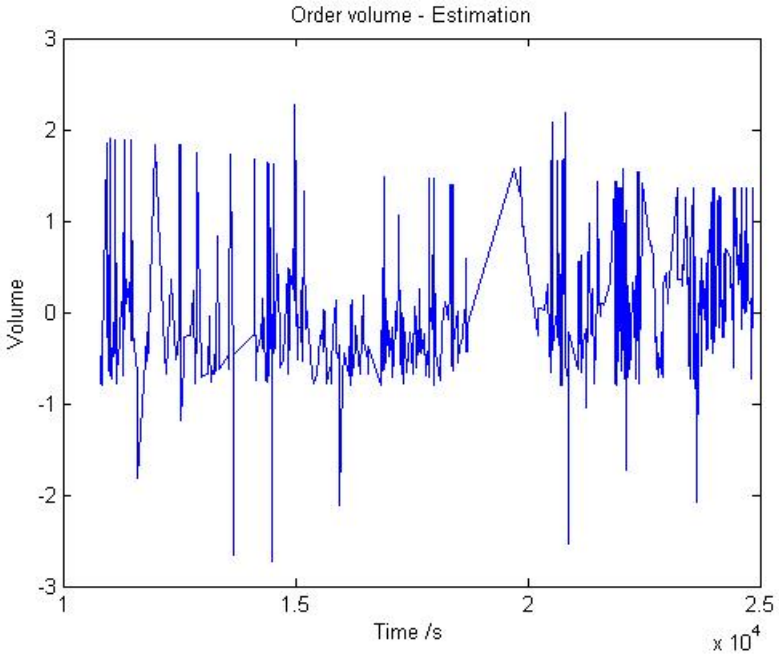

Figure 6.2. Order volume - Estimation data

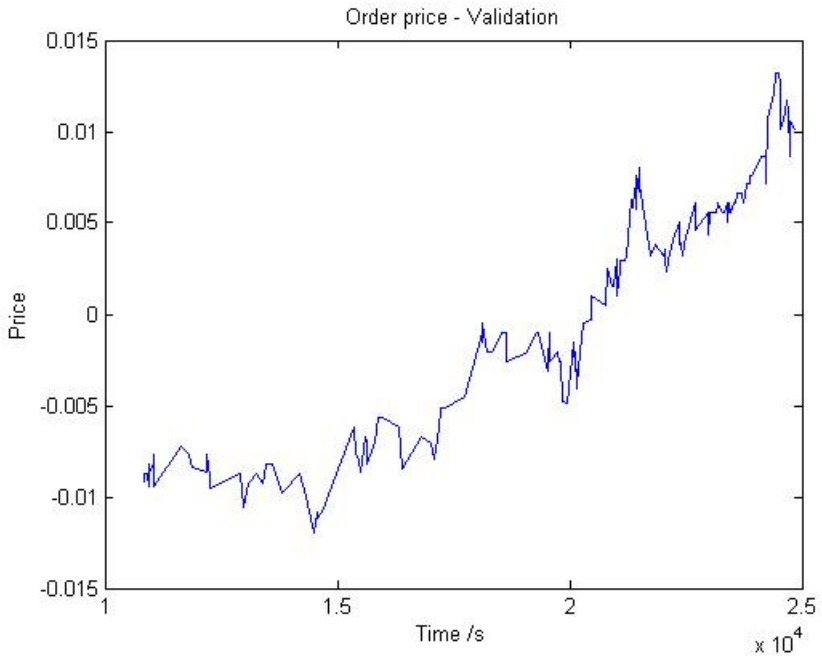

Figure 6.3. Order price - Validation data

AR-model

The AR(p)-model is a stationary autoregressive process of order p. The process depends on p former output values and is defined as:

$$\phi(B)X(t) = Z(t) \tag{6.2}$$

$$\phi(B)X(t) = X_t + \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} \tag{6.3}$$

$$Z(t) \sim N(0, \sigma^2_{WN}) \tag{6.4}$$

The Yule-Walker estimation of an autoregressive process of order p is defined as: [Brockwell 2002]

$$\hat{\phi} = (\hat{\phi}_1, ..., \hat{\phi}_p)' = \hat{R}_p^{-1}\hat{\rho}_p \tag{6.5}$$

and

$$\hat{\sigma}^2 = \hat{\gamma}(0)[1 - \hat{\rho}_p'\hat{R}_p^{-1}\hat{\rho}_p], \tag{6.6}$$

where

$$\hat{\rho}_p = (\hat{\rho}(1), ..., \hat{\rho}(p))' = \frac{\hat{\gamma}_p}{\hat{\rho}(0)} \tag{6.7}$$

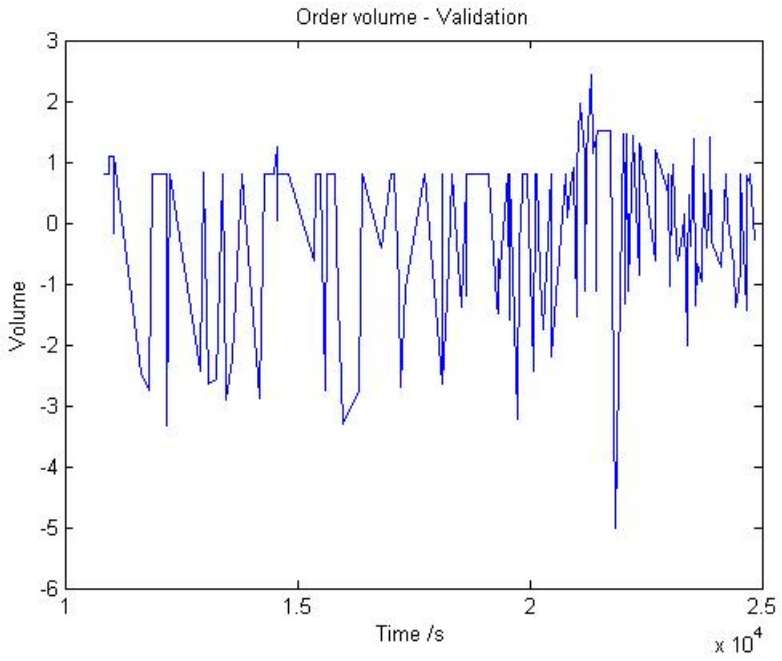

Figure 6.4. Order volume - Validation data

6.3.3 Residual Analysis

There are a number of ways to validate a selected model. The residual analysis focuses on which conclusions that can be drawn from the prediction errors, when a stochastic process is the underlying process. From the analysis, one want to show that one or more of the following statements are approximately fullfilled for the estimated model: [Brockwell 2002]

- The residuals are uncorrelated, which is equivalent to $\{Z_t\} \sim \text{WN}(0,\sigma^2)$.
- The residuals are independent if $\{Z_t\} \sim \text{IID}(0,\sigma^2)$.
- The residuals are normally distributed, as in $\{Z_t\} \sim \text{N}(0,\sigma^2)$

The tests that are used for the analysis of the residuals are presented in section 6.3.4 - 6.3.8.

6.3.4 Plot of $\{\hat{R}_t, t = 1, ..., n\}$ against time

Plots of the rescaled residuals versus time. The rescaled residuals should corresponds to that of a white noise sequence for the estimated model. Trends and nonconstant variation are analyzed, simply by noticing the behavior of the process. The plots are displayed with confidence levels ± 1.96. If at least 5% of the rescaled

residuals lies outside the bounds or if some values are exceeding the accepted limits very much, this might be a sign of a non-gaussian noise. Otherwise gaussian noise can be assumed as a starting point. There is no major conclusion that can be made from the plots. It is more a way to analyse extreme behaviors, if further analysis is worthwhile or not. [Brockwell 2002]

6.3.5 Histogram of $\{\hat{R}_t, t = 1, ..., n\}$

Histogram of the rescaled residuals. For the estimated model to be valid, the histogram should be centered around zero. If the histogram is not centered at origo, the model may not be appropriate as an approximation of the benchmark data. If the histogram have a shape that resembles with that of a normal distribution, with mean 0 and variance 1, one can suspect that the prediction error is normally distributed. This is a rough approximation about the distribution of the residuals and is only valid if the data really has a normal distribution. [Brockwell 2002]

6.3.6 QQ-plots

Quantile-Quantile plots, also refered to as QQ-plots, compares the quantiles between the benchmark data and an arbitary probability function. QQ-plots is a way to visualize the resemblances between the residuals and the distribution one suspects that residuals originates from. For example, a QQ-plots of the rescaled residuals against that of a t-distribution and/or that of a normal distribution gives an indication if the noise is normally distributed.

The QQ-plot should form a straight line with slope one for the probability distribution functions to be alike. The presentation of the QQ-plots is done in comparison with a straight line, with slope one, in the same figure. If the line coincides with QQ-plot further analysis is motivated. One major problem with the QQ-plot is that if the sample do not include that many values in a given intervall, the plot is going to be skewed.

The QQ-plot is a good tool for strengthening the thesis that the sample could have been generated from the distribution function, but the opposite is not true. The QQ-plot is defined as: [Hult, Lindskog 2007]

$$\left\{ \left(X_{k,n}, F^{-1}\left(\frac{n-k+1}{n+1} \right) \right) : k = 1, ..., n \right\}, \tag{6.8}$$

where $X_{k,n}$ is the ordered sample number k of a total number of n. F^{-1} is the inverse of distribution F, which the sample is compared to.

6.3.7 Plot of the Autocorrelation function

The analysis of the autocorrelation function is done by checking if the autocorrelation function is inside the limits of a confidence interval. Not more than 5% of

the observed residuals should fall outside the bonds of $\pm 1.96/\sqrt{n}$ for the model to be appropriate. Otherwise the estimated model should be replaced by some other model that fits the observation values better. One can suspect white noise when most of the values is lying inside the confidence levels. [Brockwell 2002]

6.3.8 Test of randomness and distribution test

Test of randomness is done to validate whether or not the residuals are independent identically distributed random variables. The distribution tests are utilized to confirm or reject the hypothesis that the sample data originates from the distribution they are tested against. The tests used for this purpose are:

Portmanteu test of randomness

The statistic to test the hypothesis that the sample is independent identically distributed is

$$Q = n \sum_{j=1}^{h} \hat{\rho}^2(j), \tag{6.9}$$

where n is the total number of observations. The statistic is the sum of squares of the sample autocorrelation function, with elements $\hat{\rho}(j)$. The theory behind the test is that if the random variables, $(X_1, ..., X_h)$, are N(0,1)-distributed, then it follows that the sum $\sum_{j=1}^{h} X_i^2$ is $\chi^2(h)$-distributed, with h degrees of freedom.

For the test to be valid, the random variables are assumed to originates from a normal distribution, with mean zero and variance one.

The rescaled residuals, under the assumption that the estimated model is correct, is approximately N(0,1)-distributed.

The statistic Q is asymptotically $\chi^2(h)$-distributed. The null hypothesis about the sample of the residuals is that the sample is a independent identically distributed (IID) sequence of normal distributed random variables. The confidence level of the test is denoted α.

The null hypothesis is accepted at confidence level α if $Q < \chi^2_{1-\alpha}(h)$ and rejected otherwise. $\chi^2_{1-\alpha}(h)$ is the $1 - \alpha$ quantile of the chi-squared distribution with h degrees of freedom. [Brockwell 2002] [Ljung 1987]

Ljung and Box Portmanteau test of randomness

Ljung and Box Portmanteau test of randomness is a modification of the Portmanteau test, named after Greta M. Ljung and George E. P. Box, which is more suitable for chi-square approximations with h degrees of freedom. The statistic is defined

as:

$$Q_{Ljung-Box} = n(n+2)\sum_{j=1}^{h}\hat{\rho}^2(j)/(n-j), \tag{6.10}$$

which is asymptotically chi-squared distributed with h degrees of freedom. The null hypothesis is defined as the null hypothesis for the regular Portmanteau test. [Brockwell 2002] [Ljung 1987]

Jarque-Bera test

The Jarque-Bera test is done in order to check if the sample is normal distributed. The statistic is defined as:

$$Q_{Jargue-Bera} = n[m_3^2/(6m_2^3) + (m_4/m_2^3 - 3)^2/24], \tag{6.11}$$

where $m_r = \sum_{j=1}^{n}(X_j - \bar{X})^r/n$ and n equals the total number of observations.

If $\{X_t\} \sim$ IID N(μ, σ^2), then $Q_{Jargue-Bera}$ is asymptotically $\chi^2(2)$-distributed. The null hypothesis about X_t is defined as the assumption that $\{X_t\} \sim$ IID N(μ, σ^2). The test is rejected at confidence level α if $Q_{Jargue-Bera} > \chi^2_{1-\alpha}(2)$. Rejecting the null hypothesis of the Jarque-Bera test indicates that the observed residuals are not gaussian.

Two-sample Kolmogorov-Smirnov test

The two-sample Kolmogorov-Smirnov test is a distribution test. The hypothesis that the sample originates from a certain distribution is tested.

The null hypothesis of the two-sample Kolmogorov-Smirnov test states that the sample is generated from the probability distribution function that one wants to test the sample against. Rejecting the null hypothesis implies that the equivalence between the sample and the tested distribution is not significant at the confidence level of α =0.05. The test statistic is defined as:

$$Q_{K-S} = max(|F_1(x) - F_2(x)|), \tag{6.12}$$

where $F_1(x)$ is the propotion of F_1^{-1} values less than or equal to x. $F_2(x)$ is the propotion of F_2^{-1} values less than or equal to x.

6.3.9 Fitting an AR(1)-model to order price

The estimated AR(1)-process is presented in equation (6.13), with the corresponding one-step prediction plot in figure 6.5. The mean value and some interesting quantiles are presented in table 6.1.

$$X_t - 0.9682X_{t-1} = Z_t \tag{6.13}$$

data sample		N(0,1)
5% quantile:	-1.7313	-1.6449
10% quantile:	-1.3491	-1.2816
Mean value:	0.0298	0
Median:	0.0823	0
90% quantile:	1.4459	1.2816
95% quantile:	1.6645	1.6449

Table 6.1. Mean value and 5%, 10%,50%, 90% and 95% quantiles for the rescaled residuals and corresponding values for the N(0,1)-distribution.

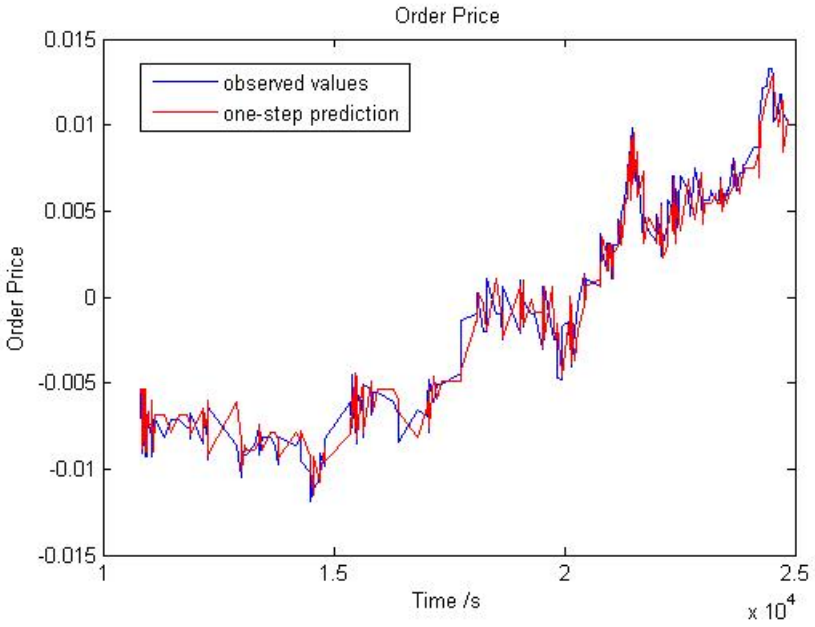

Figure 6.5. Plot of order price versus time

The forthcoming analysis is made for the order price. The same analysis is done both for order price of a bid order order aswell as for a ask order.

If one compares the figures in table 6.1, the mean value and the median are close to zero. The median is a little skewed to positive values. The quantiles for the data sample resemble with that of a normal distribution. The reason for a comparision against the normal distribution because the gaussian distribution is the most common distribution to start with. Table 6.4 gives a good approximation of the data sample, where the values are centered and the length of the quantiles can tell how

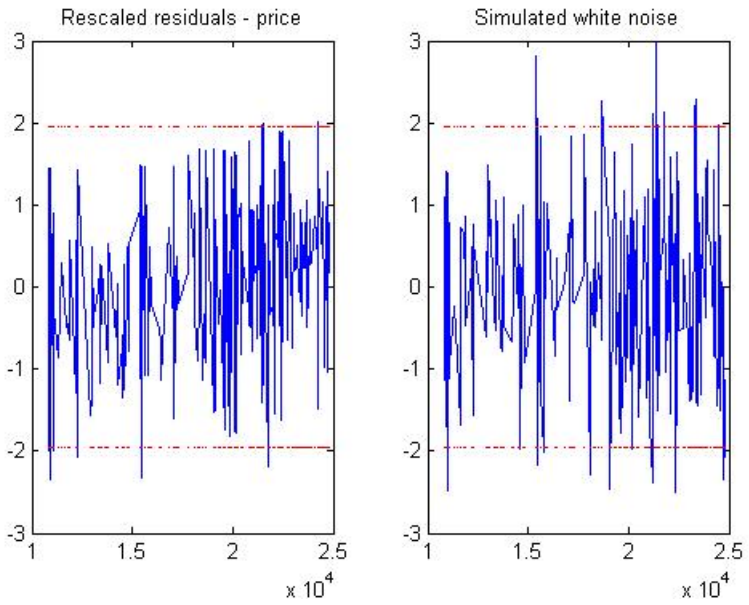

Figure 6.6. Plot of the rescaled residuals of order price versus time

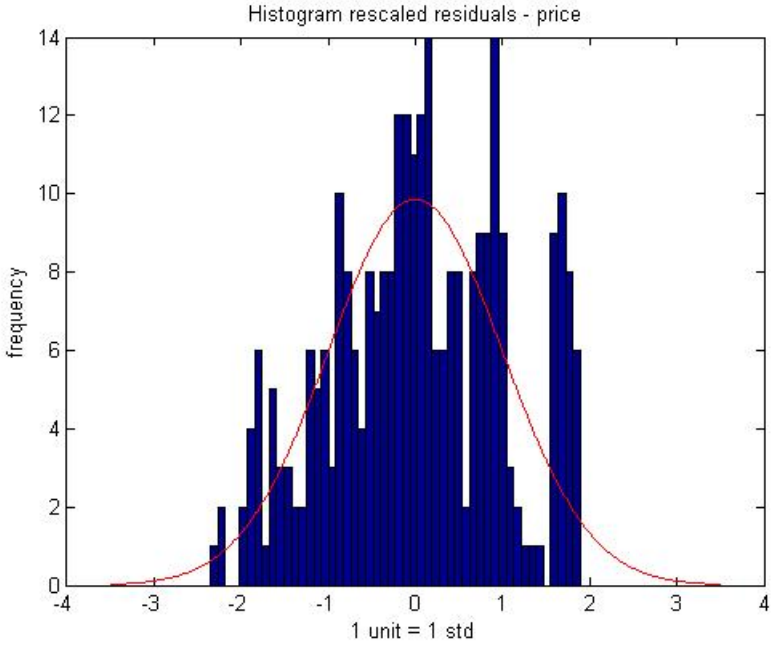

Figure 6.7. Histogram of the rescaled residuals to order price.

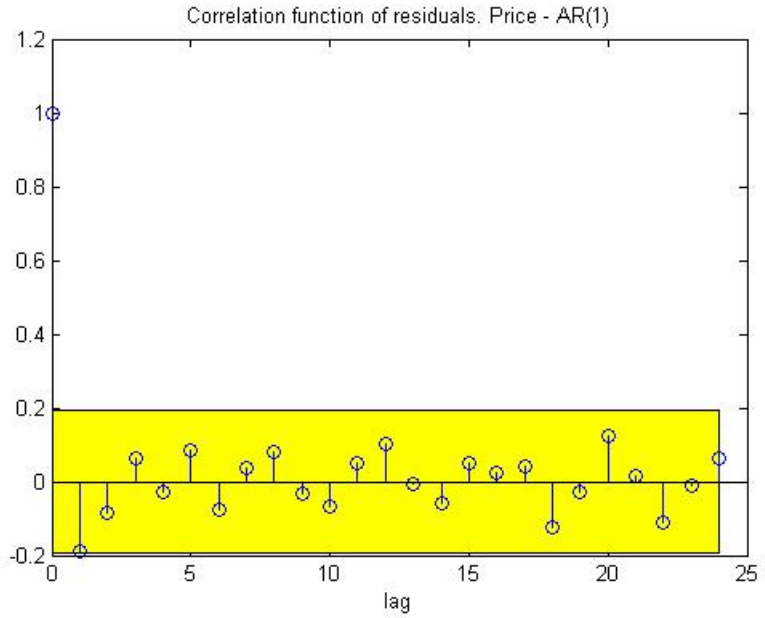

Figure 6.8. Autocorrelation function of the rescaled residuals of order price

the data sample is distributed.

The one step prediction, figure 6.5 is catching up the fluctuations of the data samples. The estimated model seems to predict the upcoming values well.

The residual analysis starts with a glance at the left plot of the rescaled residuals in figure 6.6. One can see that a few values exceed the confidence limit at ± 1.96, but by comparing with the right figure, which is simulated, there is still a good fit according to that the residuals could be a white noise sequence. There are no major differencies between the figures, which indicates that the estimated model is appropriate. If more than 5% of the peaks are outside the confidence interval one can not assume normally distributed residuals.

No major conclusions can be made of the residual plot. The first distribution to test the rescaled residuals towards will be the normal distribution.

The distribution of the rescaled residuals is visualized through a histogram, figure 6.7. The most important information that can be gathered from a histogram is if the estimated model is an appropriate one and if the shape of the histogram can imply which distribution the rescaled residuals could be a sample from.

The histogram is clearly centered at origo. Table 6.1 gives the mean value of the

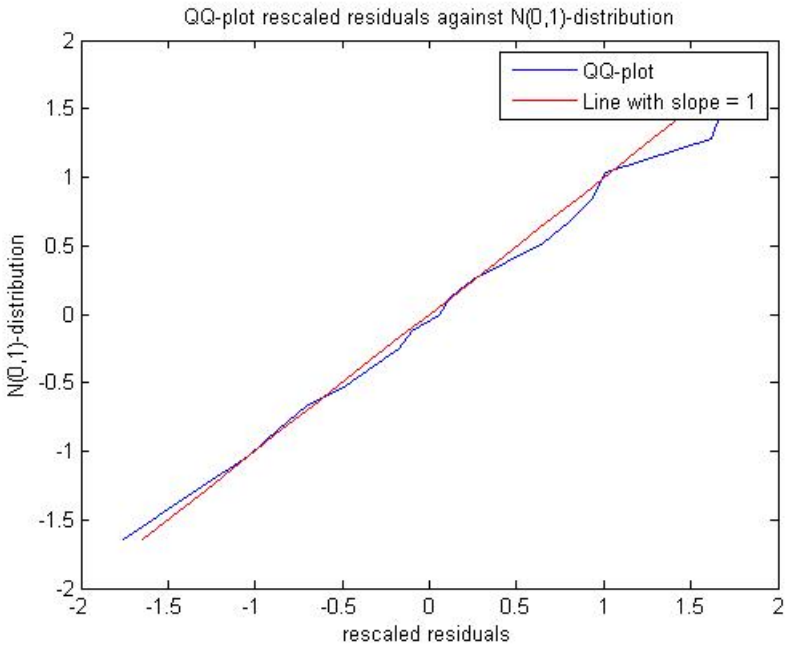

Figure 6.9. QQ-plot of the rescaled residuals of order price versus a normal distribution.

Test	p-Value	Value of test statistic	Critical Value	Reject H_0
Jarque-Bera	0.0703	4.9340	5.7667	NO
Kolmogorov-Smirnov	1	-	-	NO
Portmanteau	-	37.4549	31.4104	YES
Ljung-Box Portmanteau	0.01*	160.6376	31.4104	YES
Runstest	0.01*	-	-	YES

*Smaller than the presented p-values.

Table 6.2. Results of the tests of randomness and the distribution tests of order price

residuals, 0.0298. As stated in the previous section, a histogram centered at origo indicates an appropriate model for the sample. Some values that are not centered at origo are very frequent, but the large mass implies a gaussian shape.

The shape of the histogram is reminding of a normal distribution with mean zero and variance equal to one, due to the fact that most values are inside the limits of $\pm$ 2 standard deviations. The model seems to be appropriate and so far the normal distribution can not be rejected as the underlying stochastic process that drives the

order price.

The autocorrelation function implies whether the model is appropriate or not. If the values fall inside the confidence interval of $\pm 1.96/\sqrt{n}$, the underlying stochastic process could be assumed to be a white noise sequence. The autocorrelation function of the rescaled residuals are illustrated in figure 6.8. Most of the values are well inside the confidence interval. Even though, the value at lag one is extremely over the permitted value and several other values is on the border of confidence interval. The conclusion is that the model is fairly inappropriate and that there probably exists another model that fits the data more accurate.

The shape of the histogram, figure 6.7, implies normal distributed residuals, aswell as the figures presented in table 6.1. The quantiles between the sample and the normal distribution are very much alike. To get a closer look at the resemblence between the data sample and the normal distribution a quantile-quantile plot , QQ-plot, is presented in figure 6.9. The QQ-plot fits the straight line well.

So far the analysis has included illustration tools for getting a feeling of which distribution that seems to be appropriate for more analytically analysis. The analytically tests were described in section 6.3.3.

First of all, the estimated model is not the best model available nor is it based on a certain model validation theory. The important conlusion to make is whether or not the model is useable in terms of giving an accurate one-step prediction and determine the properties of the underlying stochastic process. The conclusions based on the underlying stochastic process must be relevant and true.

The analytical distribution tests are computed by comparing the sample data against a normal distribution. Both the Jarque-Bera test and the Kolmogorov-Smirnov test do not reject the null hypothesis that the data sample comes from a normal distribution. This means that there is no significant difference between the residuals and the gaussian distribution. The Jarque-Berra test statistic equals 0.0703, which is close to the permitted value at 0.05. The underlying stochastic process will be defined as:

$$Z_t \sim \mathrm{N}(0, \sigma^2),\ \sigma^2 = 0.0053, \tag{6.14}$$

where σ^2 is estimated with the Yule-Walker method.

The three tests of randomness will show whether or not the residuals are independent identical distributed random variables. All tests of randomness reject the null hypothesis that the residuals are IID-noise, table 6.4.

data sample		N(0,1)
5% quantile:	-2.0923	-1.6449
10% quantile:	-1.4294	-1.2816
Mean value:	-0.0035	0
Median:	0.4418	0
90% quantile:	1.0215	1.2816
95% quantile:	1.1047	1.6449

Table 6.3. Mean value and 5%, 10%,50%, 90% and 95% quantiles for the rescaled residuals and corresponding values for the N(0,1)-distribution.

The autoregressive model of order one fits the benchmark data. There are probably more suitable models that fit the benchmark data with more accuracy. The question is still how much the autocorrelation function's bad fit influences the simulation.

The same analysis, as stated in the beginning of the section, was made for order price of an ask order and for a bid order. The results were similar. Both models had residuals with a normal distribution.

6.3.10 Fitting an AR(1)-model for order volume

The estimated AR(1)-process is in equation (6.15). The same procedure as in section 6.3.10 is applied to analyze the order volume. The one-step prediction is shown in figure 6.10 and the mean value and the quantiles in table 6.3.

$$X_t - 0.2464X_{t-1} = Z_t \tag{6.15}$$

The order volume plot, 6.10, is very noisy. This can be seen by looking directly at the plot. The one step prediction does not follow the data sample at all, even though it is catching up some small trends. The one-step prediction is plotted without any underlying stochastic process, which by looking at equation (6.15) explains the mismatch between the benchmark data and the one-step prediction.

The underlying stochastic process, if the estimated model is appropriate, probably has a large variance. By the figures in table 6.3, one can notice that the median is positive, 0.4418, and the quantiles at 5% and 95% are -2.0923 and 1.1047. This gives that equally amount of samples are in the span of 0.4418-(-2.0923)=2.5341 to the left of the median as in 1.1047-0.4418=0.6629 to the right of the median. This indicates a skewness towards the positive values. The resemblance with the normal distribution is poor, as far as table 6.3 goes.

The residual analysis starts with a comparison of the rescaled residuals against

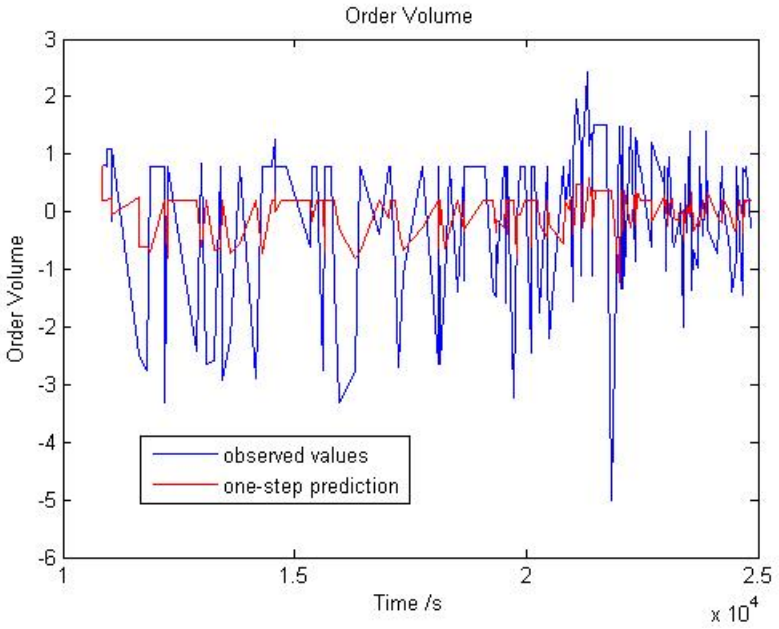

Figure 6.10. Plot of order volume versus time

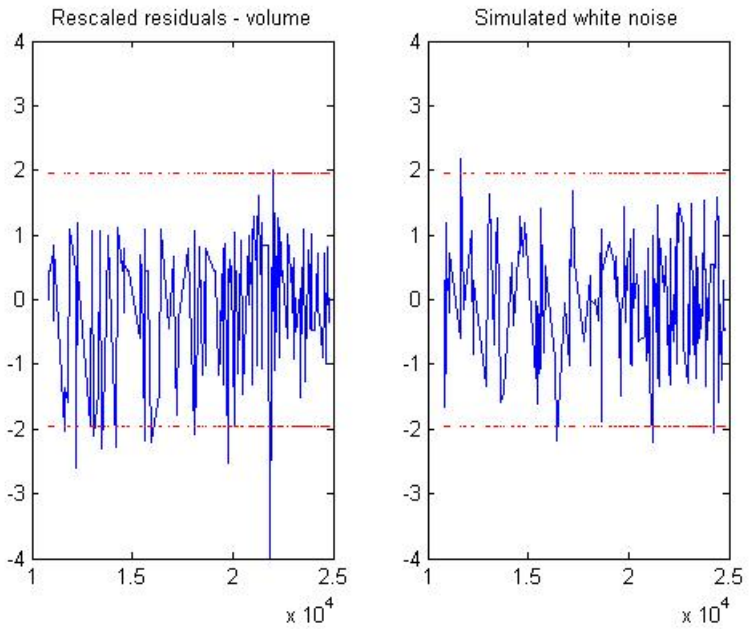

Figure 6.11. Plot of the rescaled residuals versus time.

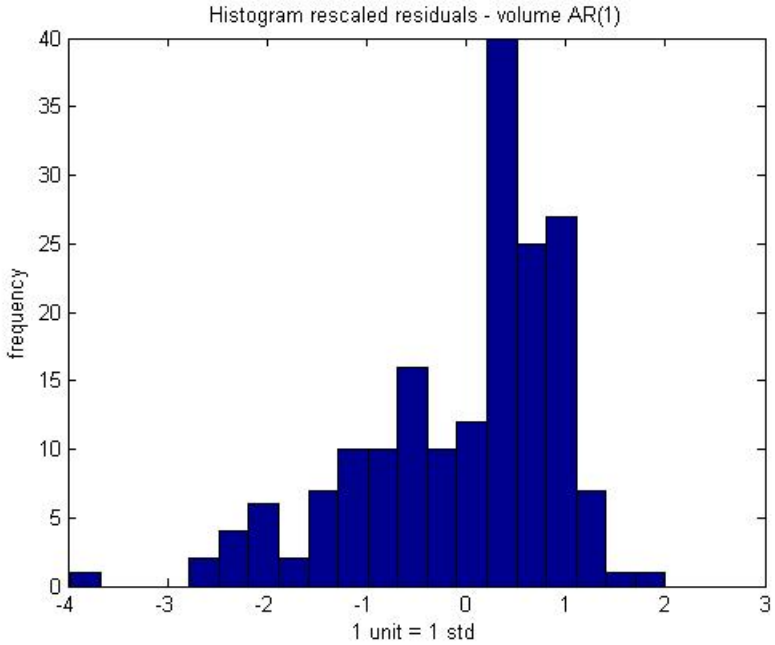

Figure 6.12. Histogram of the rescaled residuals of order volume.

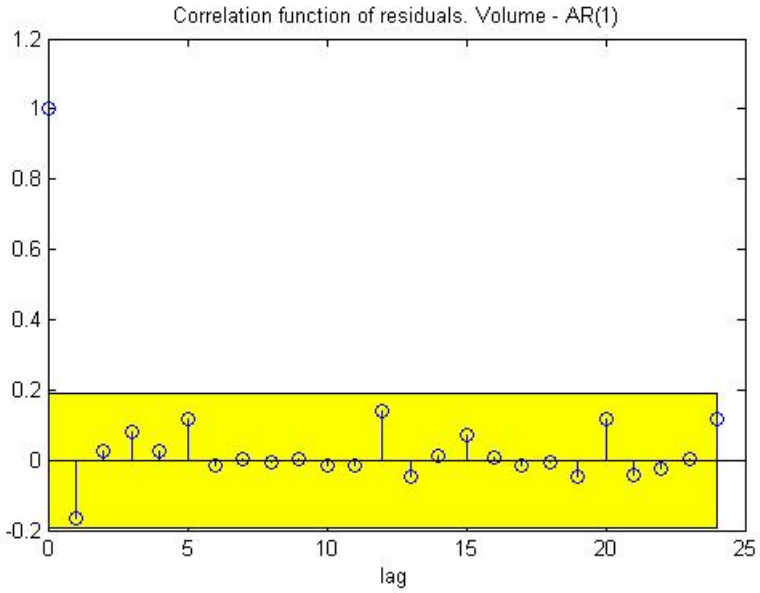

Figure 6.13. Autocorrelation function of the rescaled residuals of order volume.

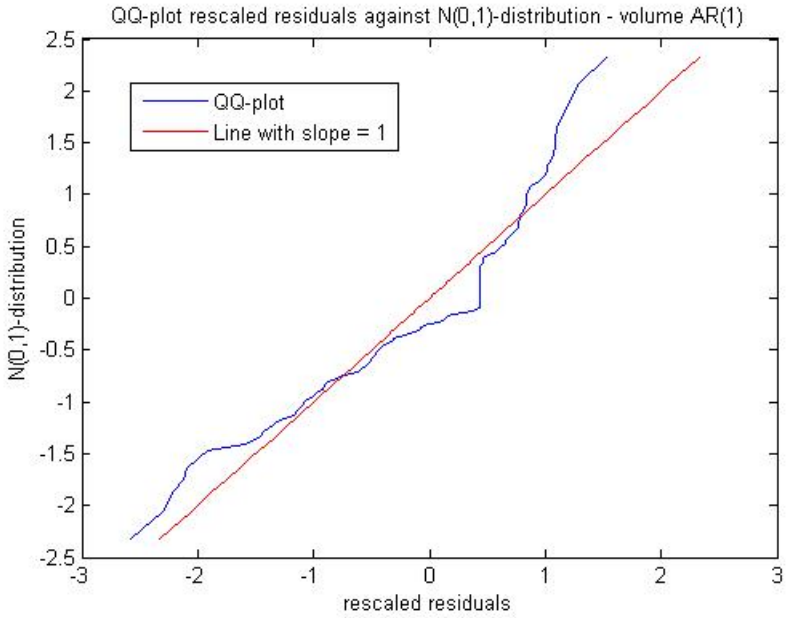

Figure 6.14. QQ-plot of the rescaled residuals of order volume versus a normal distributiom.

a simulated white noise sequence, figure 6.11. There are no major differences between the sequences, the data sample could be a white noise sequence. No obvious trends or cycles are apparent.

For the model to be appropriate the histogram, 6.12, needs to be centered at origo. The mean value is -0.0035, which is close to zero and the most of the values in the histogram are centered at origo, even though it is not quite obvious. The resemblances in shape between a normal distribution and the data sample is absent. One can suspect that the residuals have another distribution than Gaussian, which will be shown by the QQ-plot and the analytical analysis.

The autocorrelation function, 6.13, is lying inside the confidence interval at $\pm 1.96/\sqrt{n}$, even though some values are close to the border. This implies that the estimated model could be regarded as an appropriate model and that the underlying stochastic process may be a white noise sequence.

The QQ-plot, 6.14, confirm the proposition that the residuals do not originate from a normal distribution. It differs alot from the straight line, that is showing the theoretical values for an ideal match between the sample data and the normal distribution.

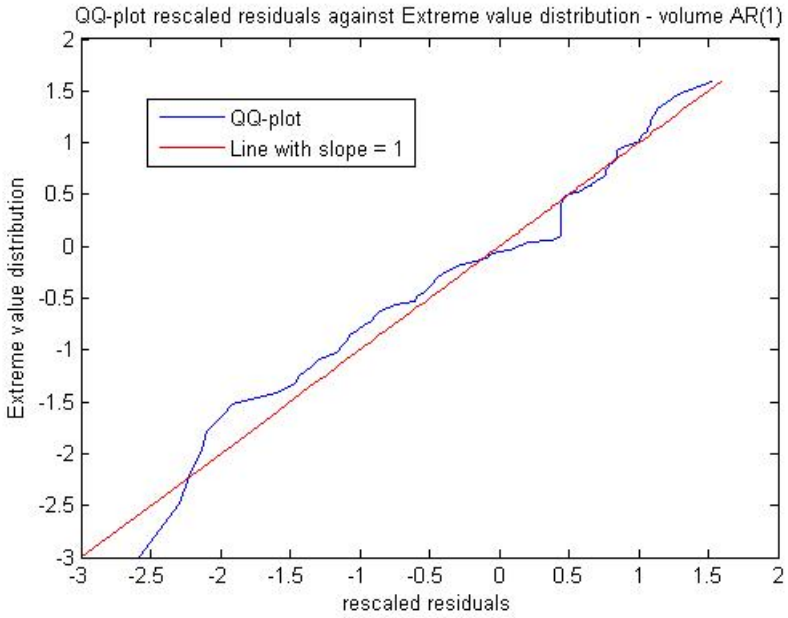

Figure 6.15. QQ-plot of the rescaled residuals of order volume versus an extreme value distributiom.

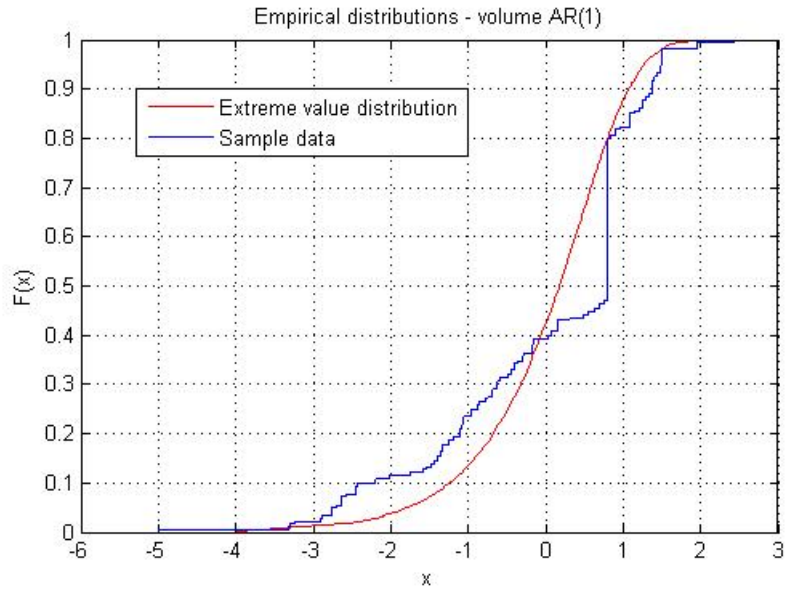

Figure 6.16. Empirical distribution of the rescaled residuals of order volume versus the extreme value distribution.

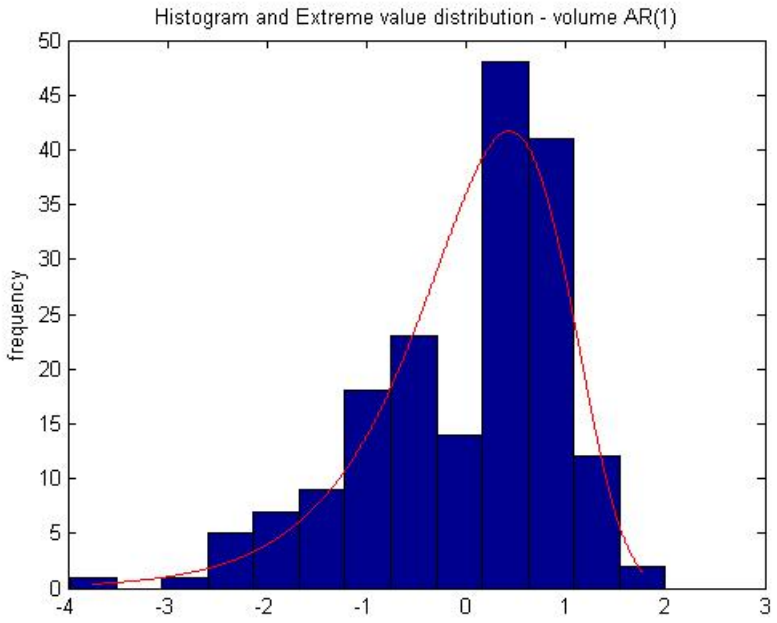

Figure 6.17. Histogram and probability distribution function of the extreme value distribution of the rescaled residuals of order volume.

Test	p-Value	Value of test statistic	Critical Value	Reject H_0
Jarque-Bera	0.01*	34.5128	5.6506	YES
Kolmogorov-Smirnov	0.0195	-	-	YES
Portmanteau	-	20.4429	31.4104	NO
Ljung-Box Portmanteau	0.5827	18.0715	31.4104	NO
Runstest	1	-	-	NO

*Smaller than the presented p-values.

Table 6.4. Results of tests of randomness and normal distribution of the rescaled residuals.

Test	p-Value	Value of test statistic	Critical Value	Reject H_0
Two-sample Kolmogorov-Smirnov	0.0963	-	-	NO

Table 6.5. Results of tests against Extreme value distribution of the rescaled residuals.

The analytical tests against a normal distribution, table 6.4, reject the null hypothesis that the residuals and the simulated normally distributed sample is derived from the same distribution. This is shown by the Kolmogorov-Smirnov test and by the Jarque-Bera test, which both have p-values that implies a significant difference.

The tests of randomness do not reject the null hypothesis that the residuals are IID-noise. The test statistics in all three tests of randomness are assumed to be normally distributed, which means that a comparision between the statistics and normal distribution is made. The accuracy of the test may be questionable because of that. Maybe the autocorrelation function should be weighted in instead, which implies non-correlated residuals. Further analysis of the residuals are needed to be able to determine which distribution the residuals may be originated from.

As stated above, the histogram, figure 6.12, illustrates a skewed distribution, weighted towards the positive values.

One distribution that have these properties is the Extreme value distribution, defined as:[Mathworld Wolfram]

$$f(x) = \frac{e^{(\alpha-x)/\beta - e^{(\alpha-x)/b}}}{\beta}, \tag{6.16}$$

where α is the location parameter and β is the scale parameter. $f(x)$ is the probability density function of the extreme value distribution.

To get a feeling of how well the residuals are fitting the extreme value distribution, the empirical distributions between the data sample and the theoretical values are compared to each other in figure 6.16. There is a decent fit in shape. Their values may differ, but the shape follows the distributions sufficient.

The next step is to illustrate the QQ-plot in figure 6.15. The QQ-plot differs from the straight line a little bit less than against a normal distribution, especially at the smaller and larger quantiles. The mismatch of the smaller and the larger quantiles are still obvious.

The analytical test, Two-sample Kolmogorov-Smirnov test, against an extreme value distribution, table 6.5, is not rejected at the confidence level of 5%. The p-value is 0.0963, which implies that there is no significance between the sample data and the extreme value distribution.

Finally, the fitted distribution is presented together with the histogram over the residuals in figure 6.17. The curve fits the histogram sufficiency well. The shapes are very much alike and the peak is centered at the same value. The underlying stochastic process of order volume is defined as:

	Data sample	True values
$\hat{\mu}$:	6.3514	
$\hat{\sigma}$:	1.3991	
5% quantile:	95	57.4
10% quantile:	105.2	95.4
Mean value:	1312.3	1525.5
Median:	489	573.3
90% quantile:	3633	3443.9
95% quantile:	4121	5725.3

Table 6.6. Mean value and 5%, 10%,50%, 90% and 95% quantiles for the data sample and true values generated from a Log-normal distribution, with the estimated parameters.

$$Z_t \sim EV(0, \beta^2), \tag{6.17}$$

where β^2 is the variance of Z_t.

This model is based on the assumption that the order volume is time dependent. One can argue if this really is the case. Due to the fact that the market participants are filtered and not regarded when analysing the data, the argument about a time independent order volume could be valid. If the market participants were taken into account, there could have been a dependece between an actor and the quantity at which they trade. But the current case is that the order volume does not have an identity, nothing that states who places the order. If there was a mark at each order, one could analyze individual frequency at which each trader makes their trades and through that get a dependence between the order volume and time. This is only true if it can be shown that different actors place different quantities in orders, which means that maybe some actors have more liquidity than others. In section 6.3.11 a time-invariant model for order volume is presented.

6.3.11 Fitting an independent identically distributed noise model for order volume

The procedure of determine a probability density function is somewhat similar to that in section 6.3.10, where a normal distribution was inappropriate to describing the residuals. The reason to analyze the volume as IID-noise is that it is not obvious that the best way to simulate order volume is by fitting the benchmark data to an AR(1)-process. For a market place system that treats different orders from market participants, the order volume may as well be dependent on which actor that is placing the order, rather than which volume last inserted order had. This section gives another approach of simulating an actor. Instead of simulating the order vol-

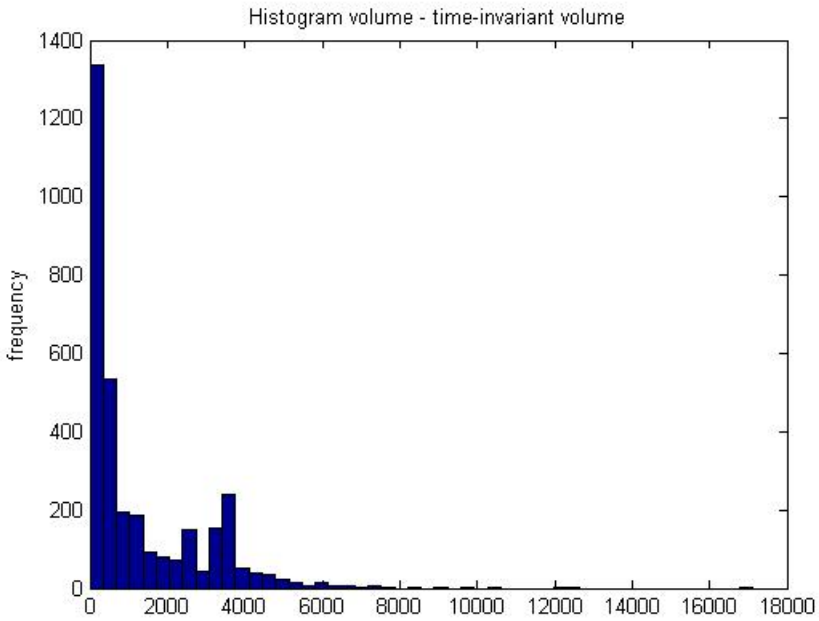

Figure 6.18. Histogram of order volume for the IID-model.

ume according to an AR(1)-process as in section 6.3.10, the order volume can be simulated from the probability distribution function determined in this section.

Several distributions were tested, such as Gamma distribution, Exponential distribution and Log-normal distribution. The only one presented here is Log-normal distribution, hence the Log-normal distribution was the best fitting distribution for this sample data. Due to the assumption that the order volume is both IID-noise and price independent, several orderbooks were used to gather the data sample. The last assumption is not self-explanatory. It could be argued that a low stock-price allow the market participants to buy a larger amount of stocks, which is neglected here. The assumption is more based on trends in the market, such as low relative price on a stock implies a larger order volume.

The estimated parameters are illustrated in table 6.6, together with quantiles similar to the analysis of the AR(1)-process. The probability density function is defined as: [Wikipedia Log-Normal distribution]

$$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{(ln(x)-\mu)^2}{2\sigma^2}};\ \text{for } x > 0, \tag{6.18}$$

where μ is the mean value and σ is the standard deviation of the corresponding normal distribution.

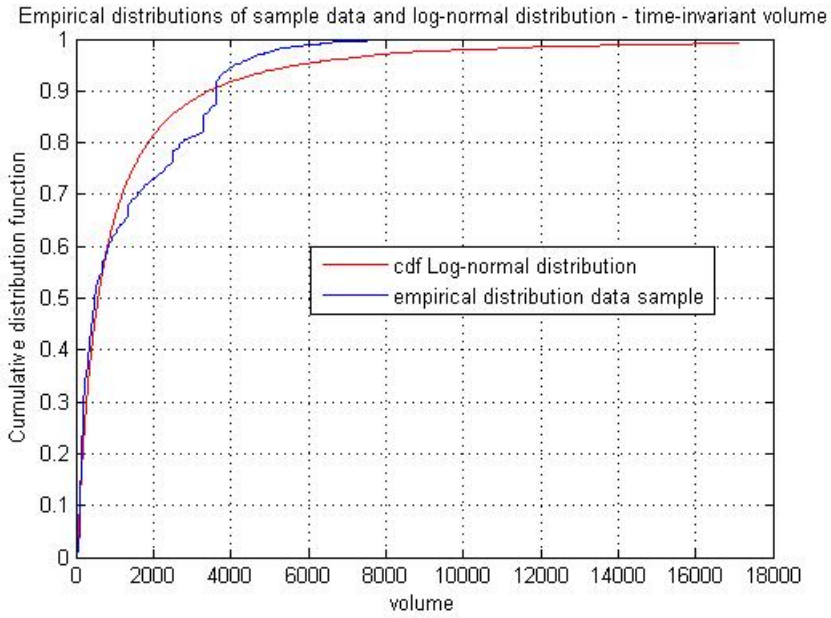

Figure 6.19. Empirical distribution of order volume against a Log-normal distribution for the IID-model

The analysis starts with a histogram, 6.18. The purpose is give a schematic picture of the shape of the distribution. As stated above, the Log-normal distribution has the same properties that are visualized in the histogram. It has the highest probability at lower values and decreases rapidly as the values increases. The log-normal distribution is characterized by a long tail with small probabilities for large values.

Some tests were done on orderbooks with small mean values of the order volume aswell as on orderbooks with large mean values of order volume. The estimated parameters were very much alike with fairly small differencies. This implies that the order volumes have small impact on estimated model and that the log-normal distribution is very flexible.

The empirical distribution of the data sample is compared with the Log-normal cumulative distribution function to the left in figure 6.19. One can notice that log-normal cdf does not increase to one as fast as the data sample. For smaller volumes the estimated distribution is more accurat.

As shown in the histogram, large order volumes are rare. This creates a problem which is well illustrated in the QQ-plot in figure 6.20. One can fairly see any resemblance between the straight line and the line that includes the sample data.

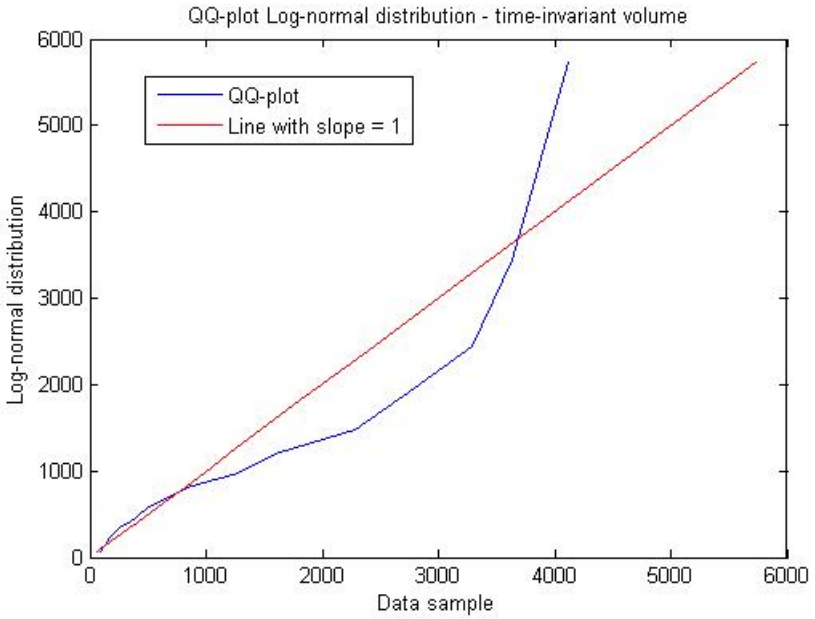

Figure 6.20. QQ-plot of order volume against the Log-normal distribution for the IID-model

The QQ-plot divides the data sample into quantiles. If the data sample do not have that many large values, the quantiles act in a skewed manner. For example, the 95%-quantile for the data sample has the value of 4121 and for the same quantile for the true distribution, with the estimated parameters, the value is 5725. That is a huge difference. On the other hand, the 5%-quantile give the values 95 respectively 57 from the previous mentioned example. The median for the data sample is 489 and for the true distribution it is 573. For clarification, the QQ-plot illustrates values from zero to values larger than 5000. That is a large domain of sample data and if the median have values equal to around 500, that would mean that as many values are gathered below 500 as above.

The mismatch could depend on the lack of large values in the benchmark data. The quantiles in the sample data "consumes" the few large values, which makes the QQ-plot inaccurat.

Further analysis is done analytically through the Kolmogorov-Smirnov test against a log-Normal distribution, which gave the p-value according to table 6.7. The test does not reject the null hypothesis that the data sample originates from a Log-normal distribution.

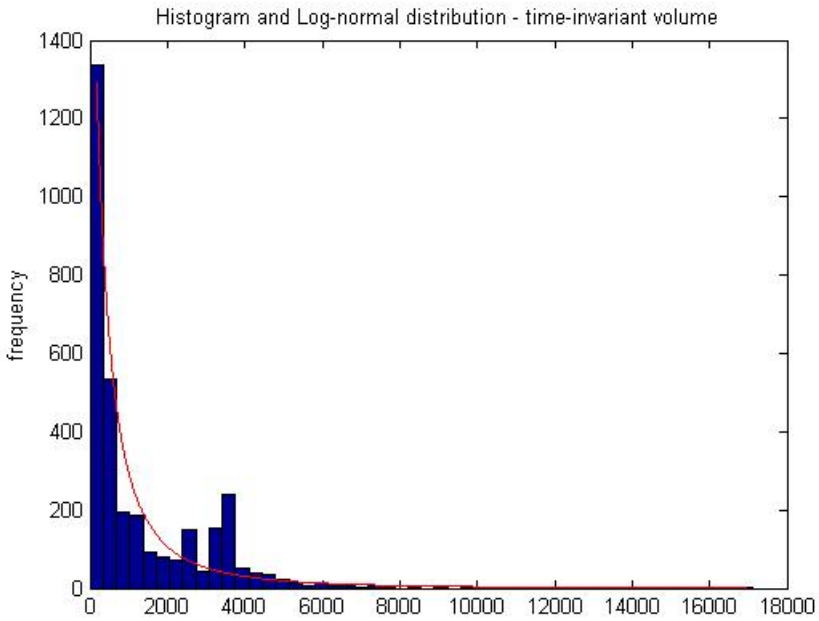

Figure 6.21. Histogram and probability distribution function of the Log-normal distribution for order volume for the IID-model

Test	p-Value	Value of test statistic	Critical Value	Reject H_0
Kolmogorov-Smirnov	0.9998	-	-	NO

Table 6.7. Results of tests versus a Log-normal distribution of order volume.

6.4 Implementation of the models

The aim of this master thesis is to make a realistic input domain for random testing in order to validate the functionalities of a version of TRADExpress.

The benchmark data was filtered and prepared for a mathematical analysis. The analysis was made by fitting an autoregressive process of order one to the benchmark data of order price.

For order volume, both a time independent analysis and a time dependent analysis were made. As for order price, the time dependent model for order volume is adapted to an AR(1)-process. The probability distribution function for the time independent assumption is approximated by a log-normal distribution. These results

are implemented in this section, along with the simulation tools that are being used in order to perform random tests.

6.4.1 Market participants

The market participants in this implementation are a number of traders. The traders were implemented as all functionality in one, which means that each trader is able to do everything that a trader should do. Each trader has the ability to put every single order available in the simulation, both buy and sell orders. The trader is executed as a thread, which makes it possible for them to trade independently of each other. They have a "listener" implemented, that keeps track of the last traded price on the market for each orderbook. The orders are placed stochastically, with a frequency according to the intensity of a Poisson process. The procedure of each trader is that they login into the system, place a pre-defined number of orders and finally logout.

Order price

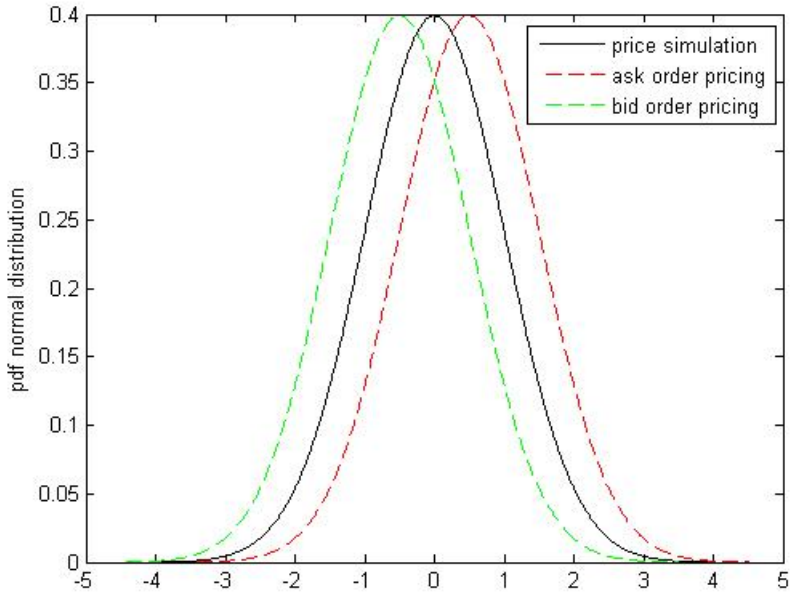

Figure 6.22. Order pricing

When simulating the order price, the trader calls on an AR(1)-process, which determines the forthcoming market price. Whether it is a buy order or a sell order, the pricing of the order is determined from the current market. A buy order is

Type of order	Subcondition	Probability
Limit	None	1/4
	Iceberg	3/20
	Fill-Or-Kill	1/20
	Fill-And-Kill	1/20
Peg-Order	None	1/4
	Iceberg	3/20
	Fill-Or-Kill	1/20
	Fill-And-Kill	1/20

Table 6.8. Probabilities when simulating order types.

settled as $X_t - \frac{\sigma^2}{2}$ and a sell order is priced as $X_t + \frac{\sigma^2}{2}$. A schematic figure of how the orders are settled is illustrated in figure 6.22.

X_t, the order price, is simulated from:

$$X_t = 0.9682X_{t-1} + Z_t;\ Z_t \sim \mathrm{N}(0, \sigma^2);\ \sigma^2 = 0.0053 \tag{6.19}$$

X_{t-1} is the reference price that corresponds to the last inserted order in the current orderbook and Z_t is the underlying stochastic process, or equivalently the prediction error.

Order volume

The order volume was implemented according to the independent identically distributed model, section 6.3.11. Simulations are generated from a Log-normal distribution with $\mu = 6.5634$ and $\sigma = 1.3863$, where μ and σ corresponds to the parameter in a normal distribution. For this purpose, the Log-normal distribution is a smooth distribution to do simulations from, due to the definition that excludes negative sample values.

For very large orders the transparancies of the orders are dark. This is valid for orders worth over five millions Swedish kronors and they are not visible to the public eye.

Placing an order

The orders available for the traders are Limit orders and Peg-orders with probability $p = 0.5$ for each order. The frequency of each order are calculated from the historical trading data, by counting the occurencies of each order as if they were placed according to an uniform distribution with probability $p = \frac{1}{\text{Total number of trades}}$.

One could interpret it like a sample from an empirical distribution of Limit orders

and Peg-orders. When a Limit order or a Peg-order is assigned, another simulation is made whether the order should remain as it is or if it should have properties according to that of an Iceberg-, Fill-or-Kill or Fill-and-Kill order. The probabilities that are assigned to each particular order are presented in table 6.8.

For large orders, as stated above, the total order is not visible for the public eye. When the limit of five millions swedish kronor is exceeded, the order is generated according to a Dark order or a Minimum volume order. They have probability $p = 0.5$ each.

When these attributes are set, the order is inserted in the orderbook. The orderbook chosen is simulated pseudo-randomly from an uniform distribution with probability $p = \frac{1}{\text{Number of Orderbooks}}$ for each orderbook. The number of orderbooks is the amount of orderbooks implemented when testing.

The orderbook keeps track of the different tradable instruments, one for each orderbook. In the implementation the orderbook is used for scaling the amount of orders to each orderbook. The total number of orders during one day is calculated as the total amount of orders into the trading system. If one orderbook is used, all orders will be placed in that orderbook. If three orderbooks are implemented, one third of the total amount of orders will be placed in each orderbook.

The same goes for the traders. The more traders implemented, the longer the time between inserted orders from each trader is. The amount of orders per second is still the same.

The orders are inserted with pseudo-random frequency, generated from a Poisson-process with three different intensities.

The first intensity corresponds to the opening hours for the market place, the first two and a half hours. The intensity of the Poisson process was calculated to $\lambda_1 = 31.6717$ orders per second.

The second time interval is during the day. The calculated intensity is $\lambda_2 = 23.2533$ orders per second.

The third time interval corresponds to the two last hours of the trading day. The intensity of that interval is $\lambda_3 = 40.9400$.

The intensity is calculated as the total amount of orders inserted to the system per second. The time between each order done by the trader is calculated as the inverse of the randomly generated expected value for orders per second. This figure is multiplied by the number of traders implemented.

Analysis of results

The results of a Random test can be analyzed in several different ways using some of the Oracle strategies. To test the functionality of a market place system, the No Oracle strategy is suitable and can expose rare event errors that can cause the system to crash, which is of major importance. The No oracle strategy was used for the simulation of this master thesis.

6.5 Simulation and Results

6.5.1 Simulation

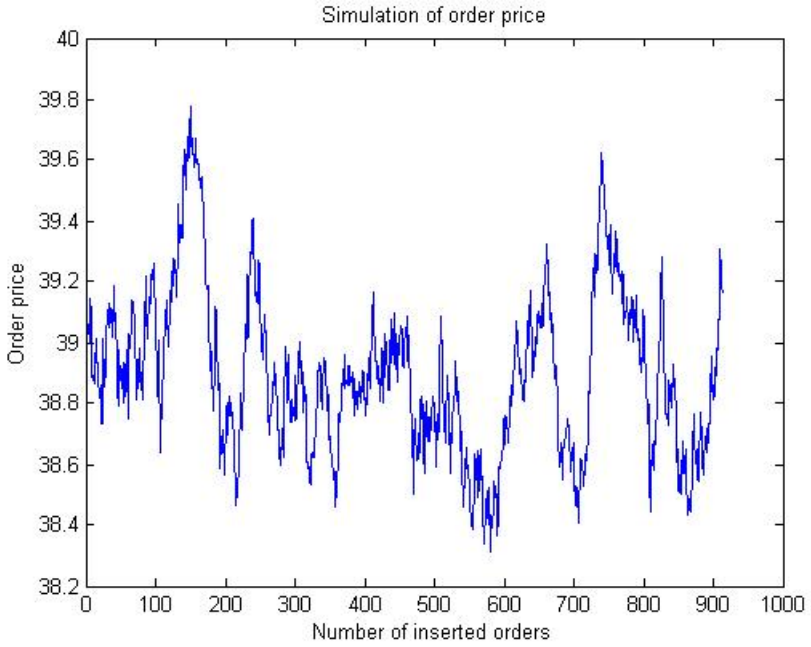

Figure 6.23. Simulation of order price

The simulations were executed with ten traders, placing their orders in three different orderbooks. Several simulations were executed from the 27th of April to the 15th of May. The system under test was a newly developed trading system, which went public the 8th of May.

The simulated model is illustrated in figure 6.23. The plot describes the pattern from which the trades generate their orders. The traders were placing their orders according to the actual price described in section 6.4.1.

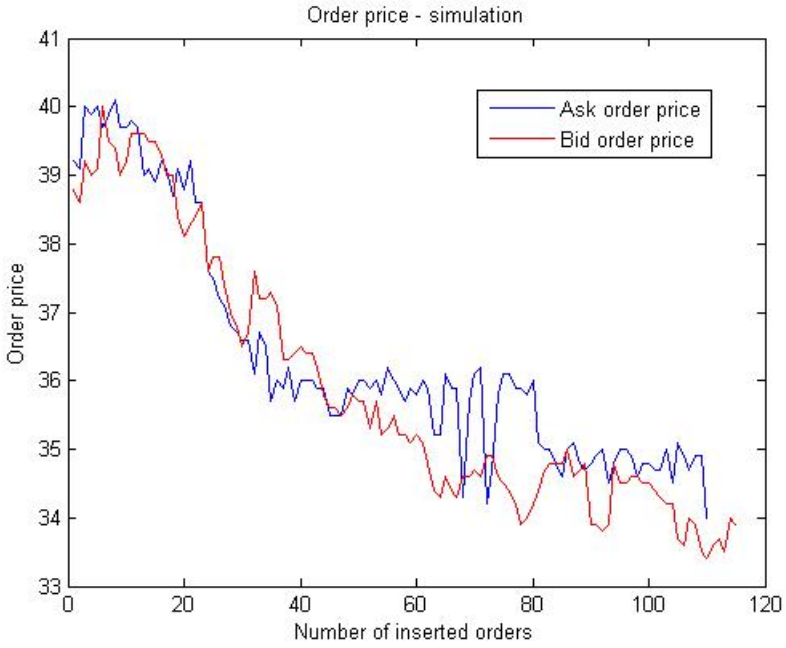

Figure 6.24. Simulation of order price for bid and ask order

An illustration of order insertion is shown in figure 6.24. The plot is schematic to illustrate the differencies. The variance of the process was manipulated so that the differencies are more obvious. The most important to notice in the figure is that the ask order insertion is higher than the bid order inserted. One early simulation, shown in figure 6.25, illustrates the opposite situation. In that case, every inserted order gave an execution of the order. The real situation that corresponds to that behavior would be that every trader that wants to buy the stock as expensively as possible and every trader that wants to place a sell order will sell as cheaply as possible, which is a contradiction. The reason for the behavior was that the order prices were updated separately according to if it was an ask order or a bid order that triggered the trade. In the later version, one index curve was simulated in order to determine the price from which the orders were priced, which gave a more realistic trading pattern.

The time series represented in figure 6.23 was analyzed aswell, just to verify the process. The same analysis as in section 6.3.9 was made to the simulated data. The mean value and corresponding quantiles are shown in table 6.9. The similarity between the simulation and the normal distribution with mean value zero and standard deviation one is obvious, regarding the spread of the sample.

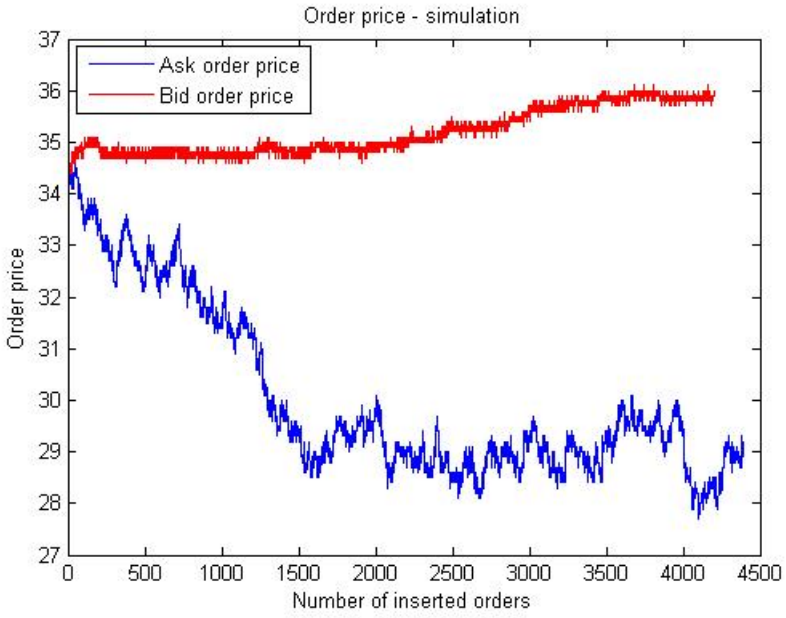

Figure 6.25. Simulation of order price for bid and ask order, bad implementation.

data sample		N(0,1)
5% quantile:	-1.6669	-1.6449
10% quantile:	-1.2706	-1.2816
Mean value:	0.0015	0
Median:	-0.0238	0
90% quantile:	1.2636	1.2816
95% quantile:	1.6430	1.6449

Table 6.9. Mean value and 5%, 10%,50%, 90% and 95% quantiles for the rescaled residuals and corresponding values for the N(0,1)-distribution.

The Histogram, figure 6.26, illustrates the correspondence even more. The histogram is centered at origo, with a shape very similar to that of a normal distribution. The curve that follows the histogram is the true normal probability function with $\mu = 0$ and $\sigma = 1$.

Finally, the QQ-plot is presented in figure 6.27. The quantile-quantile curve follows the straight line very well. The two sample Smirnov-Kolmogorov test did not reject the null hypothesis that the sample data is generated from a normal distribution.

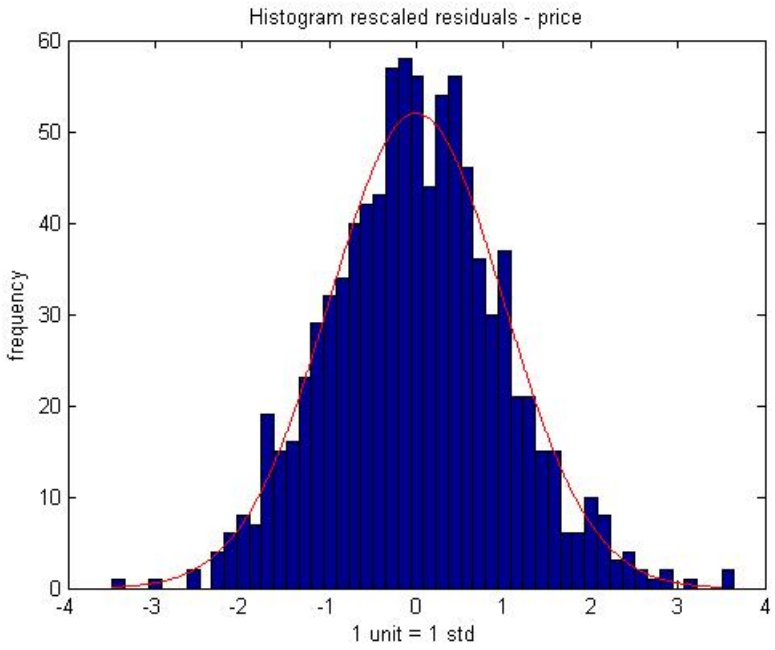

Figure 6.26. Histogram of rescaled residuals for the simulation of order price

The simulation of the trading pattern corresponds to the model implemented. The implemented model on the other hand is based on benchmark data from a real trading day, which implies that the simulation is appropriate to simulate a trading day of stock exchange markets.

The sample data from the simulation of the order volume is illustarted by a histogram in figure 6.28. The curve in the figure represents the true Log-normal probability function with $\mu = 6.5634$ and $\sigma = 1.3863$. The order volume is simulated according to the estimated model, which corresponds to an approximation of the real benchmark data. The probabilities seem to be accurate with respect to the real trading data.

6.5.2 Results of Random Testing

The random tests did not expose any failures. No major critical errors that should have caused the system to crash were found. The major reason for this is probably that the tests were executed on a system that has already been tested due to functionality, at least for tests as complex as the implemented random tests. The system under test has been tested frequently and very rigidly for a long time, so no major errors were expected to be found. Mean time to failure can for obvious

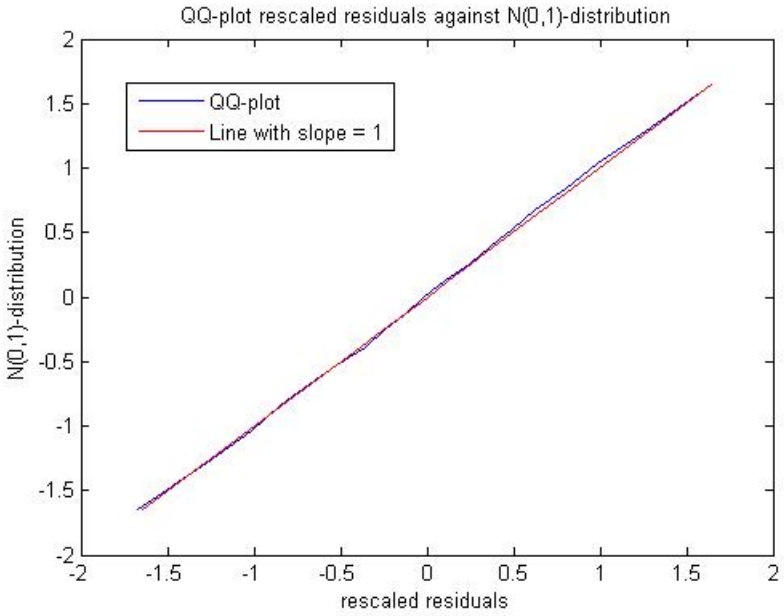

Figure 6.27. Histogram of rescaled residuals for the simulation of order price

reasons not be measured.

The functionality can still be analyzed. The implemented market participants traded independently with each other. While testing, the tester can follow the market in real-time and make sure that orders are inserted and executed. If any problems occur under the simulation, a message of the error is printed and saved in a textfile. An example of the most frequently showed error-message was:

Tue May 12 09:21:38 CEST 2009 CFT8 Order cannot be priced since there is nothing to base pricing off

The messages states that a Peg-order can not be inserted, because it can not be priced according to any price in the current orderbook. This is not a failure, just a message that the system under test will not allow an order that can not be priced in the orderbook. Similar messages were found for orders that have hidden volumes, where for example the open order quantity was larger than the total order.

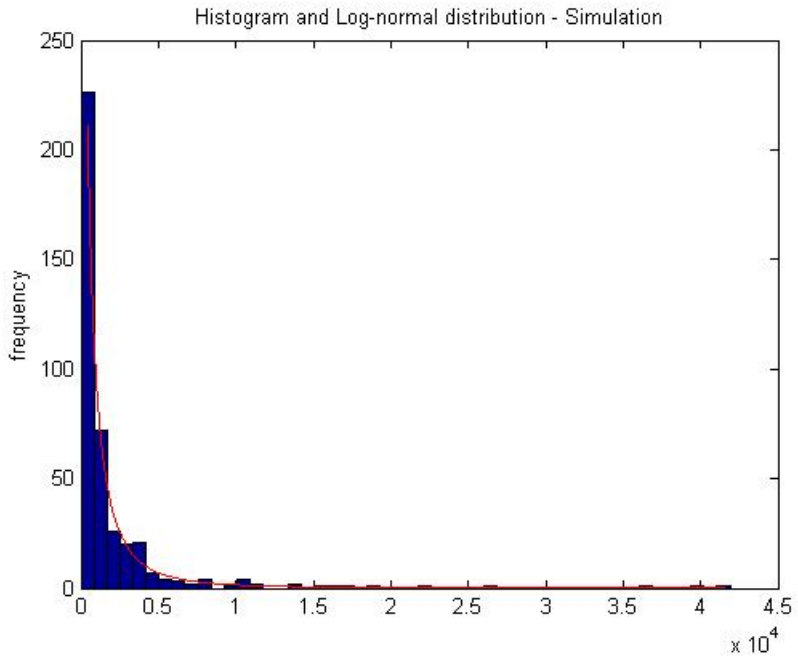

Figure 6.28. Histogram of order volume for the simulation and the probability distribution function of the Log-normal distribution.

Chapter 7

Discussion

The aim for this master project was to analyze real benchmark trading data in order to fit an appropriate model to the data. The purpose was to get a realistic input domain for random testing and test the software's functionality. The simulation did show similar behaviors to that of a real trading day. The simulated results are adequate for the purpose. The random tests did not expose any failures with the implemented model. In that sense another model may have been at least equally efficient in finding errors. The currently used model would have performed just aswell as the simulated model.

Due to the fact that the model is based on real trading data, the random tests is still a good way of testing the functionalities of the system. The magnitude of which the benchmark data is analyzed can be extended further, according to the complexity of order insertion. There are a lot of order combinations that can be analyzed in order to increase the complexity of the orderbook's order depth. This could be a way of trigger rare events that cause the system to crash. Instead of using the No oracle strategy, one could use some other oracle strategy in order to analyze the results more efficient. Some minor failures that not causes the system to crash may have been found in that way.

The AR(1)-processes were not validated by any model validation theory, such as AIC, Akaike Information Criterion, or BIC, Bayesian Information Criterion, which may have improved the resembles between the model and the benchmark data. It would probably not have exposed more failures, but the model's accuracy may have been improved. Some other AR(p)-processes and ARMA(p, q)-processes were fitted to the benchmark data. They are not included in the rapport and were never analyzed as rigorously as the implemented models.

The functionalities of the software under test were tested. The software performed the functions according to the specifications. The software that was tested is in use and should manage to work under normal circumstances, which this test has

shown that it can. Normal circumstance is equivalent to a trading pattern from an ordinary trading day, which is the case for the implemented model. In order to find rare events that causes failures, the tests need to be executed often under very long periods of time and the model may have to be broader, so that rare events are triggered more often.

7.1 Conclusion

The hypothesis of the master thesis was to find a model that works as a realistic input domain for random testing. The implemented model, under the given restrictions given by the test framework, simulates a real trading day well. Random testing is efficient when testing a software's functionality.

7.2 Further work

This master thesis is just the beginning for creating a framework for random testing. As stated in chapter 7, the complexity of the orderbook can be analyzed further in order to cover more of the possible orders and the pattern at which they are traded. A more extensive model can be implemented in order to cover more market participants and make the trading schedule even more realistic. The implemented model could be extended with a master project that develops an Oracle that covers the features of realistic trading events and the corresponding events that are not permitted.

Appendix A

References

[Albers 2004], Albers, Marcel; Jonker, Catholijn M.; Karami, Mehrzad; Treur, Jan, *Agent Models and Different User Ontologies for an Electronic Market Place*, DOI 10.1007/s10115-002-0092-3, Springer-Verlag London Ltd., 2004.

[Brockwell 2002], Brockwell, Peter J.; Davis, Richard A., *Introduction to Time Series and Forecasting*, Second Edition, ISBN 978-0-387-95351-9, Springer 2002.

[Cinnober 2009], *TRADExpress Trading System - Business Functions and Architecture, Cinnober Finacial Technologies*, 2009.

[Edvardsson 1999], Edvardsson, Jon, *Department of Computer and Information Science, Linköping University*, Sweden, 1999.

[Fu 2003], Fu, Rui, *Automatic Test Suite Generation Based on Test Oracles*, The University of Western Ontario, London, 2003.

[Grottke 2001], Grottke, Michael, *Software Reliability Model Study*, IST-1999-55017, PETS, 2001.

[Gundemark 2005], Gundemark, Johan, *Random Tests in a Market Place System*, Uppsala University, Department of Information Technology, Uppsala, Sweden, 2005.

[Gut 1995], Gut, Allan, *An Intermediate Course in Probability*, ISBN 0-387-94507-5, Springer, 1995.

[Gutjahr 1999], Gutjahr, Walter J., *Partition Testing vs. Random Testing: The Influence of Uncertainty*, Department of Statistics, Operations Research and Computer Science, University of Vienna, Vienna, Austria, 1999.

[Hamlet 1994], Hamlet, Richard, *Random Testing, Encyclopedia of Software En-*

gineering, pages 970-978, Wiley, 1994.

[Hamlet 2006], Hamlet, Richard, *When Only Random Testing Will Do*, Portland State University, Portland, USA, 2006.

[Hoffman 1998], Hoffman, Douglas, *A Taxonomy for Test Oracles*, Software Quality Methods, LLC., 1998.

[Hoffman 1999], Hoffman, Douglas, *Heuristic Test oracles*, STQE Magazine March/April 1999, (pages 29-32).

[Hoffman 2001], Hoffman, Douglas, *Using Oracles in Test Automation*, Software Quality Methods, http://www.softwarequalitymethods.com/Papers/Auto, 2001.

[Hoffman 2003], Hoffman, Douglas, *Mutating Automated Tests*, STAR East 2000, Software Quality Methods, LLC., 2000.

[Hult, Lindskog 2007], Hult, Henrik; Lindskog, Filip, *Mathematical Modeling and Statistical Methods for Risk Management*, Lecture Notes, Department of Mathematical Statistics, Royal Institute of Technology, Stockholm, Sweden, 2007.

[Höjeberg 2007], Höjeberg, Noah, *Random Tests in a Trading System*, Royal Institute of Technology, Department of Computer Science and Communication, Stockholm, Sweden, 2007.

[Kaner 2003], Kaner, Cem; Bond, Walter P.; McGee, Pat, *High Volume Test Automation*, Florida Institute of Technology, 2003.

[Kuo 2007], Kuo, F.-C.; Chen, T.Y.; Liu, H.; Chan, W.K., *Enhancing Adaptive Random Testing in High Dimensional Input Domains*, ACM 1595934804/07/0003, Seoul, Korea, 2007.

[Ljung 1987], Ljung, Lennart, *System Identification: Theory for the user*, ISBN 0-13-881640-9, P T R Prentice Hall, 1987.

[Mayer 2006], Mayer, Johannes; Schneckenburger, Christoph, *An Empirical Analysis and Comparison of Random Testing Techniques*, Ulm University, Dept. of Applied Information, ACM 1-59593-218-6/06/0009, 2006.

[NASA 2004], *NASA Software Safety Guidebook*, 2004.

[Schauer 2006], Schauer, Philip Martin, *Market architecture of the largest stock exchanges*, Leopold-Franzens-Universität, Innsbruck, Institut für Banken und Finanzen, Fakultät für Betriebswirtschaft, 2006.

[Sellberg 2003], Sellberg, Lars-Ivar, *MEXUS-main design ideas v1.3*, 2003.

[Spillner 2007], Spillner, Andreas; Linz, Tilo; Schaefer, Hans, *Software Testing Foundations*, ISBN 978-1-933952-08-6, 2nd edition, Rocky Nook Inc., Santa Barbara, 2007.

[Sthamer 1995], Sthamer, Harmen-Hinrich, *The Automatic Generation of Software Test Data Using Genetic Algorithms*, University of Glamorgan / Prifvsgol Morgannwg, 1995.

[Wooff 2002], Wooff, David A.; Goldstein, Michael; Coolen, Frank P.A., *Bayesian Graphical Models for Software Testing*, IEEE Transactions of Software Engineering, Vol. 28, No. 5, 2002.

A.0.1 Electronical references

[Wikipedia Market], http://en.wikipedia.org/wiki/Market

[Wikipedia Marketplace], http://en.wikipedia.org/wiki/Marketplace

[Wikipedia Log-Normal distribution], http://en.wikipedia.org/wiki/Log-normal_distribution

[Mathworld Wolfram], http://mathworld.wolfram.com/ExtremeValueDistribution.html