“Scenario-based approach to backtesting trading systems”

Konstantin Savenkov (savenkov@cs.msu.su), Dennis Zhbankov,
Lomonosov Moscow State University

Abstract

In this paper we present a scenario-based system for backtesting trading strategies. The backtesting system takes a list of market orders and historical quote data as inputs and calculates a set of metrics for a given scenario.

The system incorporates simulation model of market orders and transactions execution. It is based on a mathematic model of market order execution and allows for testing systems that employ full range of modern market orders. The system analyzes market volume and therefore is capable of simulating partial execution of market orders. The mathematic model of market execution include static and operational semantic of market orders. It specifies how brokerage account affects the possibility of market order registration, how a market order transforms to a sequence of transactions (in case of partial execution) and how the transactions affect brokerage account. As the inputs are market orders, the system could be used for testing arbitrary trading system. Such a trading system could be a mechanical trading system, human trading system or even a scenario of actions for a trader, whom trading system is unknown.

Therefore, the backtesting system presented in this paper could be used for backtesting and risk analysis of mechanical trading systems, backtesting and algorithmization of non-mechanical trading systems, evaluation of a human trader’s skills and capabilities.

Also we have collected useful set of metrics that are calculated by the backtesting system based on a given scenario and historical quote data. The system is implemented as an open-source software project. This paper presents a case study of comparing three different trading strategies, which demonstrates benefits of the proposed approach to the backtesting.

Keywords: stock market, trading, backtesting, risk analysis, simulation

Introduction

The main question for all stock market players concerns defining optimal time when to buy and to sell stock. To answer this question, a player uses so-called trading strategy (or trading system) – a set of rules that define conditions for opening and closing market positions. These conditions could depend on either technical or economic indicators or events. Thus, it is very important to know how to estimate and compare efficiency of different trading strategies.

Such estimations are desired to solve a number of practical problems:

1) Development of trading strategies and algorithms,
2) Validation and risk analysis for mechanical trading systems,
3) Optimization of trading strategies,
4) Evaluation of human trader’s skills and capabilities.

The most popular approach to evaluation of trading strategies is backtesting [6] – testing the strategy on historical quote data. The existing backtesting tools are intended for evaluation of mechanical trading systems. In such backtesting tools, historical quote data are passed as inputs to the trading system, which is implemented as a computer program. Using the quote data, the trading system generates a sequence of limit buy and sell orders. Then these orders are processed by the backtesting system to estimate variation of cash equivalent value of broker’s assets over

time. During the processing, it is assumed that the traded volume is insufficient to influence the market price. The trace of the broker's account's variations is used to calculate a set of efficiency metrics for the trading strategy (maximum drawdown, payoff ratio, profit compared to buy&hold strategy, profit compared to random walk etc).

The scheme described above is employed by virtually all existing backtesting systems. However, it possesses several intrinsic flaws, which impose restrictions on applicability of backtesting and limit accuracy of the backtesting results. The flaws are enumerated below.

1. Evaluation of mechanical trading systems only.
2. Restrictions on the means of specifying of the trading strategies (internal languages with limited capabilities and low performance).
3. Limited set of supported market orders. Most of the existing backtesting tools allow for usage of simple limit orders and stop orders with a fixed price.
4. Market traded volume in historical quote data is not considered.

The backtesting approach that is proposed in this paper is free of these flaws. Inputs for the backtesting tool are historical quote data and a trading *scenario*, which is a sequence of market orders. That allows our backtesting tool to be used for evaluation of trading strategies of different nature, including evaluation of human trader's performance. As for evaluation of mechanical trading systems, that provides an ability to describe trading algorithm using any programming language and unlimited possibilities to define custom trading efficiency metrics.

The backtesting tool utilizes historical market traded volume data to assess possibility of market order's execution. That allows for simulation of partial market order execution.

The proposed approach to backtesting is based on *the formal model of market order registration and execution* (see Section 1). This model specifies all types of market orders that are available in modern electronic stock trading systems [11]. Using this formal model, the *simulation model of stock exchange* has been built. It simulates how broker issues an market order, how the order is registered (or dismissed) by a trading system and how the active marker order is executed (transformed in a sequence of transactions). The simulation model realizes a market trading loopback: the state of a brokerage account defines the broker's capability to issue market orders; executed order is transformed in a sequence of transactions; a transaction, in its turn, affects the state of the brokerage account. Variation of the brokerage account over time is used to calculate a set of efficiency metrics.

The backtesting tool presented in this paper is implemented an open source software [1].
The paper is organized as follows. In the first section, the formal model of market order registration and execution is briefly described. In the second section, we give a list of available trading efficiency metrics, describe architecture of the backtesting tool and make an example of using this tool for comparing several trading strategies. In the Conclusion, limitations of the proposed approach and further development plans are discussed.

1. The formal model of market order execution

The proposed formal model consists of two parts. The first part defines *static semantics* of the model by specifying notions of *brokerage account*, *market order* and *transaction*. The second part defines *operational semantics* of the model by formal definition of market order registration, market order execution (i.e. transforming market order in a sequence of transactions) and transaction processing. This section presents a condensed version of the formal model.

1.1 Static semantics

Basic definitions

Domains of Money (a set of all possible amounts of money) and Time (a set of all possible timestapms) are defined as subsets of positive real numbers. For the sake of simplicity, a single stock market model is considered. Therefore, a set of all possible amounts of stock is defined as a subset of integers.

$$Money \subseteq R^+, Time \subseteq R^+, Stock \subseteq Z$$

Brokerage account

Brokerage account is defined as a collection of cleared monetary assets, cleared stock assets and both monetary and stock assets, reserved for execution of active market orders (successfully registered in electronic trading system).

$$Account = Money_{CLRD} \times Stock_{CLRD} \times Money_{RZVD} \times Stock_{RZVD}$$

Transaction

Transaction corresponds to a successfully executed market order and describes atomic pair of transitions: a transition of stock from a brokerage account of a seller to a brokerage account of a buyer, and a transition of money backwards. Here, transactions are considered from the viewpoint of a backtested trading system, therefore two types of transactions are identified: purchase and sale.

$$TransactionType = \{Purchase, Sale\}$$

A domain of possible transactions is defined as a collection of transaction type, traded price, traded volume and a timestamp:

$$Transaction = TransactionType \times Money \times Stock \times Time$$

Historical quote data

Historical quote data is represented as a set of transactions, ordered by timestamps:

$$History \subseteq 2^{<Transaction>}$$

Market order

Five types of market orders are identified: market, limit, stop loss, trailing stop and take profit:

$$OrderType = \{Market, Limit, StopLoss, TakeProfit, TrailingStop\}$$

A domain of Orders if defined as a collection of order type, type of corresponding transactions, desired price, desired volume, registration timestamp and cancellation timestamp:

$$Order = OrderType \times TransactionType \times Money \times Assets \times Time \times Time$$

Order types are described in more detail in Section 1.2 “Operational semantics”.

Broker scenario
Scenario of broker’s actions is defined as a set of orders, ordered by registration timestamps:

$$Scenario \subseteq 2^{<Order>}$$

1.2 Operational semantics

In accordance to the notation introduced in the previous section, the backtesting problem reduces to evaluation of the following functions:

1. The *backtest* function projects from a product of an initial brokerage account, a broker scenario and historical quote data into a sequence of transactions over the brokerage account:
   $backtest: Scenario \times Account \times History \longrightarrow 2^{<Transaction>}$

2. A set of $metric_i$ functions, each projects a sequence of transactions in a value of *i*-th trading efficiency metric:

$$metric_i: 2^{<Transaction>} \longrightarrow R$$

A possibility of registering an order in electronic trading system depends on a state of the brokerage account, which, in its turn, depends on a result of execution of previously registered orders. That makes impossible direct (i.e. analytic) evaluation of the *backtest* function and prompts for the usage of simulation.

An algorithm for the trading session simulation is presented below. Then we define semantics of several auxiliary functions and derivation rules for the operational semantics of order registering and execution. The derivation rules are specified using a standard $\frac{cause}{effect}$ form, which means that if predicate *cause* evaluates to *true* for the current state of the model, then the rule is applicable and predicate *effect* evaluates to *true* after its application. In *effect*, elements of the model’s state that have been modified by application of the rule are marked by a stroke (‘).

1.2.1 Broker state

A state of the broker is defined as a collection of modeling time t_M, brokerage account state a, a list $active$ of registered orders, broker scenario sc and historical quote data $hist$:

$$State = Time \times Account \times 2^{Order} \times Scenario \times History.$$

1.2.2 Order registration semantics

Possibility of registering an order is defined by a function *valid*:

$$valid: Account \times Order \times Time \times History \longrightarrow \{true, false\}$$

An order *o* is *valid* to be registered at a given time *t* if the amount of cleared monetary assets on a brokerage account *a* is sufficient to be reserved for the order execution, taking into account historical quote data *h* for the time *t*:

$$valid(a,o,t,h) = free(a) \geq vol(o) * price(o,t,h)$$

Here, the function $price: Order \times Time \times History \longrightarrow Money$ depends on time *t* and historical quote data *h* for the market type orders only.

Thus, operational semantics of the order registration could be defined as follows. If the model time t_M is equal to the timestamp of the first order $o = top(sc)$ from the scenario *sc*, then the order *o* is processed and removed from the scenario. In case the order *o* is valid at that moment t_M, it is added to the list of *active* orders. If *o* is a purchase order, then the corresponding amount $vol(o) * price(o,t,h)$ of cleared monetary assets $money_{CLRD}(a)$ become reserved for the execution of *o*.

$$\frac{time(o) = t_M \wedge valid(a,o,t_M,h)}{\begin{array}{l} sc' = sc \backslash o \wedge active' = active \cup o \wedge \\ money'_{CLRD}(a) = money_{CLRD}(a) - vol(o) * price(o,t,h) \wedge \\ money'_{RZVD}(a) = money_{RZVD}(a) - vol(o) * price(o,t,h) \end{array}} \vee \frac{time(o) = t_M \wedge ! valid(a,o,t_M,h)}{sc' = sc \backslash o}$$

A sale orders are treated in a similar way, but in such case the corresponding amount $vol(o)$ of cleared non-monetary assets $stock_{CLRD}(a)$ is reserved.

1.2.3 Order execution semantics

Operational semantics of the order execution defines a function that projects the order's parameters and historical quote data into a set of transactions:

$$exec: Order \times History \longrightarrow 2^{<Transaction>}$$

Function *exec* is defined as a set of derivation rules. A pair of rules is defined per each of the ten possible kinds of an order[1], one rule for the full execution of an order and another one for the partial execution. Below there is an example of operational semantics for a sell limit order[2].

Full execution of a sell limit order

Execution of a sell limit order *o* is planned (see **Section 1.2.6**) on a timestamp of such transaction h_i from historical quote data, that the timestamp is greater than the timestamp $time(o)$ of the order, and the stock price in the transaction h_i is not lesser than the price $price(o)$ stated in the order.

---

[1] A number of possible kinds of an order is defined by $|OrderType| * |TransactionType| = 5 * 2 = 10$.

[2] Sell limit order *o* is an order such that $order\_type(o) = Limit,\ transaction\_type(o) = Sale$

In case the volume of the transaction h_i is greater or equal to the volume stated in the order, the order is fully executed, and the list $exec(o)$ of transactions corresponding to the order is entailed by a transaction on a full volume of the order.

$$\frac{h_i = \inf\{h \in History | time(h) \geq t_M \wedge time(h) \geq time(o) \wedge price(h) \geq price(o)\} \wedge volume(h_i) \geq volume(o)}{exec'(o) = exec(o) \cup \{(Sell, price(o), volume(o), time(h_i))\} \wedge active' = active \backslash o}$$

Partial execution of a sell limit order

In case the volume of the transaction h_i from historical quote data is not sufficient to fully execute the order o, the transaction list $exec(o)$ is entailed by a transaction that partially executes the order. Volume of the active order o is reduced to reflect the partial execution, the new order execution event for the order is scheduled in the future.

$$\frac{h_i = \inf\{h \in History | time(h) \geq t_M \wedge time(h) \geq time(o) \wedge price(h) \geq price(o)\} \wedge volume(h_i) < volume(o)}{exec'(o) = exec(o) \cup \{(Sell, price(o), volume(h_i), time(h_i))\} \wedge volume'(o) = volume(o) - volume(h_i)}$$

1.2.4 Order recall

In case some order o has been recalled, all events corresponding to the order are erased from a list of events scheduled in the future; assets reserved for execution of the order become cleared; the order itself is removed from the list of active orders.

1.2.5 Transaction processing

When a purchase transaction $tr \in Transaction$ is processed, an amount of cleared stock $stock'_{CLRD}(a)$ on the brokerage account $a \in Account$ increases, while the money paid for the stock is subtracted from the monetary assets reserved for execution of the order corresponding to the transaction.

$$\frac{time(tr) = t_M \wedge type(tr) = Buy}{stock'_{CLRD}(a) = stock_{CLRD}(a) + stock(tr) \wedge money'_{RZVD}(a) = money_{RZVD}(a) - price(tr) * volume(tr)}$$

In a similar fashion, when a sale transaction is processed, the amount of stock reserved for the registered order is reduced, while the amount of cleared monetary assets is increased by the monetary volume of the transaction.

1.2.6 Trading session simulation

The simulation employs the discrete-event approach and identifies three types of events:

1. order registration,

2. order execution (including fulfillment of the order execution condition),
3. transaction processing (i.e. a change of the brokerage account).

The simulation algorithm is as follows:

1. Fetch the topmost order from the broker scenario sc.
2. According to the order, brokerage account and historical quote data, schedule in the future order registration event.
3. Advance the simulation time to the earliest event and process the event in the following way:
    a. **Order registration.** The possibility of order registration is checked. In case of the positive result, the order is placed in a list of active events and the order execution event is scheduled.
    b. **Order execution.** If condition of a limit order is fulfilled or a market order is registered, then the order is converted in a sequence of transactions by using rules of the operational semantics. The transactions are planned as transaction processing events in future.
    c. **Transaction processing.** When a transaction is processed, it is recorded in the history of the brokerage account changes and a new state of the brokerage account is calculated.

2. Backtesting tool prototype

The proposed approach to backtesting of trading strategies is implemented and published under open-source license [1]. The architecture of the backtesting tool is outlined below, along with a brief description of trading efficiency metrics calculated by the tool and a simple case study of trading strategy comparison.

2.1 Trading efficiency metrics

There is a number of common metrics for assessing efficiency of trading activity [2-5]. These metrics are calculated in a course of backtesting and incorporated in the backtesting report. Most of the metrics described below are implemented in some of the existing backtesting tools [12-18]. However, this set has been extended by several metrics found in research papers only [7-10].

According to the formal model of the order execution proposed above, a *trading efficiency metric* is a function that projects from a set of possible transaction sequences into a set of real numbers. However, to calculate some of the metrics it is more convenient to work with a *set of deals* instead of a transaction sequence. This set is populated by deals that close market positions. The set of deals is unambiguously constructed from a sequence of transactions using the FIFO[3] principle. Profit for a deal is calculated by subtracting price of a deal that has opened positions from the deal that closes the position.

The set of implemented efficiency metrics is divided in several groups: metrics calculated for loss-making deals only; metrics, calculated for profitable deals only; metrics calculated for the whole scenario; statistical metrics.

[3] FIFO – First In First Out.

Metrics for profitable deals

- **gross profit:** overall profit from all profitable deals,
- **adjusted gross profit:** overall profit from such profitable deals, that their profit do not exceed 3 standard deviations,
- **profit deals number:** number of profitable deals,
- **average profit deal time:** average time between opening deal and closing deal for profitable deals,
- **percent of profit deals:** percent of profitable deals among all deals,
- **max profit:** maximum profit from a profitable deal,
- **average profit:** average profit from a profitable deal,
- **max continuous profit:** maximum profit, given by consequent profitable deals,
- **max continuous profit number:** maximum length of a sequence of profitable positions;

Metrics for loss-making positions

- **gross loss:** overall loss from all loss-making deals,
- **adjusted gross loss:** overall loss from such loss-making deals, that their loss do not exceed 3 standard deviations,
- **loss deals number:** number of loss-making deals,
- **average loss deal time:** average time between opening deal and closing deal for loss-making deals,
- **percent of loss deals:** percent of loss-making deals among all deals,
- **max loss:** maximum loss from a loss-making deal,
- **average loss:** average loss from a loss-making deal,
- **max continuous loss:** maximum loss, given by consequent loss-making deals,
- **max continuous loss number:** maximum length of a sequence of loss-making positions;

Metrics for all deals

- **total income:** overall profit from both profitable and loss-making deals,
- **deals number:** a number of deals,
- **total time in testing:** time interval between the first opening deal and the last closing deal,
- **average time in market:** average time between an opening deal and the closing deal,
- **max drawdown:** maximum drawdown of equivalent monetary value of the brokerage account from its maximum value to reaching the new maximum,
- **relative drawdown:** maximum ratio of a drawdown to local maximum of equivalent monetary value of brokerage account,
- **payoff ratio:** ratio of an average profit from profitable positions to an average loss from loss-making positions,
- **profit factor:** ratio of total income to a max drawdown,
- **relative total income:** ratio of equivalent monetary value of the brokerage account after the last closing deal to the equivalent value before the first opening deal,
- **buy-and-hold profit:** ratio of the stock price after the last closing deal to the stock price before the first opening deal;

Statistical metrics

- **profit mean:** mean value of total income,
- **profit standard deviation:** standard deviation of the total income,
- **Sharp ratio:** ratio of total income to the standard deviation,
- **T-sign:** a probability to get the same relative total income using a random walk.

2.2 Architecture of the backtesting software

Software prototype of the proposed approach to backtesting is called Backtester [1] and is organized as follows.

The historical quote data are given in CSV format and describe a sequence of recorded transactions. Recorded transactions are of a special undefined type (neither purchase nor sale), since they reflect existence of both buyer and seller. Broker's scenario is represented as a text file, which describes a sequence of broker's orders.

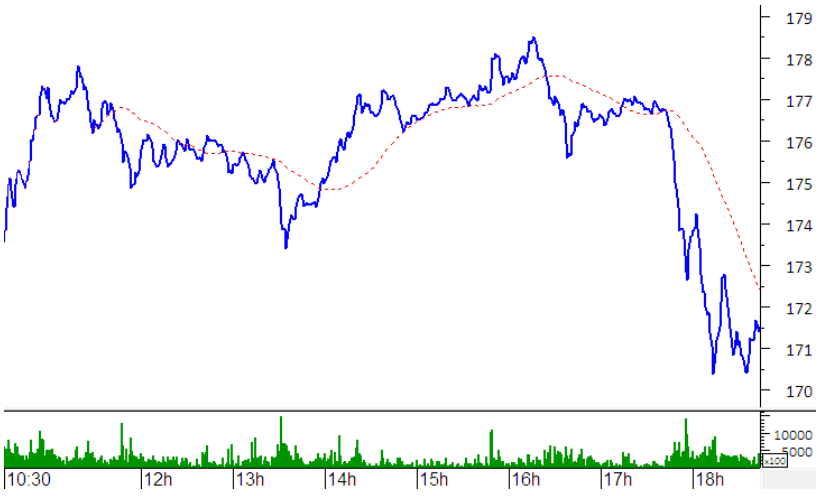
The software prototype implements the simulation model of trading session (See Section 1 "Formal model of market order execution"). Given historical quote data and a scenario, the software generates a sequence of transactions. This sequence is used to calculate a set of trading efficiency metrics (See Section 2.1). Values of the metrics are saved in a backtesting report, which is the final output of the software prototype.

2.3 Backtesting Case Study

To provide a case study for the proposed approach, several trading strategies are compared on a same period of time. The historical quote data are tick market quotes for GAZPROM stock for 5 of May, 2009 on MICEX. Each line of historical quote data file contain information about a single transaction in a CSV format (See Table 1). The date, time, price and volume of a transaction are given.

```
20090507,184416,171.60000,2
20090507,184416,171.60000,943
20090507,184416,171.55000,500
20090507,184417,171.57000,200
20090507,184417,171.60000,5402
20090507,184417,171.60000,120
20090507,184418,171.60000,250
20090507,184419,171.60000,4028
```

Table 1. A fragment of the file with the historical quote data (date, time,price and volume of a transaction).



Picture 1. Graphical representation of the historical quote data.

Graphical representation of the historical quote data is given on a Picture 1. Blue solid line denotes variation of a stock price, blue dashed line denotes the hourly average price value, green bars denote volume of transactions.

Two simple trading strategies are compared. Both strategies are based on comparison of a stock price with a hourly average value of the price. Buy orders are placed if the price become greater than its hourly average value, sell orders are placed when the price become lesser than its hourly average value. The first of the two strategies (Simple) operates only simple market orders. The second strategy uses market orders, limit orders and stop orders.

The scenarios generated according to the trading strategies are recorded in files scenarioSimple and scenarioSmart. Each line of the scenario file defines parameters of a single order. The parameters include order number, order type, transaction type, price, volume, date and time of the order validity, number of corresponding order, spread and ident.

As a result of backtesting, one backtesting report is generated per each scenario. Most important statistics and metrics from the reports are accumulated in Table 2.

Efficiency Metric Name	**Scenatio "Simple"**	**Scenario "Smart"**
Statistics		
Deals number:	6000	6000
Total income:	1894.36 (0.19%)	4545.71 (0.45%)
Buy-and-hold profit:	0.11%	0.40%
Total time in testing:	06:08:42	05:48:51
Average time in market:	00:32:55	00:24:27
Max drawdown:	1464.62	1350.00
Relative drawdown:	0.15%	0.13%
Profit mean:	0.32	0.76
Profit standard deviation:	1.03	1.26
Sharp ratio:	0.002	0.004
Payoff ratio:	14.658	7.511
Profit factor:	1.293	3.367
Profit		
Profit deals number:	1005	4000
Max continuous profit number:	999	2999
Average profit deal time:	02:21:49	00:35:49
Gross profit:	2600.52	4869.89
Adjusted gross profit:	2600.52	4869.89
Max profit:	2.67	3.50
Average profit:	2.59	1.22
Max continuous profit:	2597.80	4359.43
Percent of profit deals:	16.75%	66.67%
Loss		
Loss deals number:	4000	2000
Max continuous loss number:	2999	1999
Average loss deal time:	00:11:30	00:01:44
Gross loss:	706.12	324.20
Adjusted gross loss:	706.12	324.20
Max loss:	0.37	0.19
Average loss:	0.18	0.16
Max continuous loss:	542.75	324.03
Percent of loss deals:	66.67%	33.33%

Table 2. Values of efficiency metrics for scenarioSimple and scenarioSmart.

The results obviously demonstrate that the “Smart” scenario, which employs more advanced orders, has better profit and stability characteristics.

3. Conclusions

The model of order execution proposed in this paper formally defines all orders available for a broker on modern stock market. Operational semantics of the model define how the orders are registered and executed and allows for implementing of trading simulation model. The simulation model uses historical quote data and broker activity scenario to calculate a sequence of transactions over the brokerage account. The sequence is used to calculate various trading efficiency metrics.

To compare several trading strategies one should generate a trading scenario per strategy using rules of the strategy and the same historical trading data that would be used for backtesting. Then the scenarios and the historical quote data are passed to the backtesting software that implements the proposed approach [1].

3.1 Limitations of the order execution model

This paper presents the simplest model of stock market trading. It does not consider some high-demanded features of the modern stock market:

- margin trading,
- short trading,
- “market feedback”: usage of information about execution of the registered orders to decide whether to register new orders,
- transaction costs.

All of these limitations are not crucial and the model could be easily extended to remove them.

Fundamental features of real-world trading are beyond the scope of the model are that:

a) Registered orders affect market price,
b) Concurrent orders from multiple market players could prevent market orders from execution at the given

These flaws exist in any backtesting tool and, moreover, are intrinsic to the very approach that relies on historical quote data to evaluate trading strategy. Inaccuracy imposed by these flaws usually is balanced out using the slippage coefficient. We are going to supplement this technique by empirical estimation of the slippage coefficient using a ratio of an order to a historically traded volume. Besides that, the proposed approach is free from the third common backtesting problem, which is related to inaccuracy due to historical traded volume is not sufficient to fully execute an order.

3.2 Further work

Most of the limitations described above are in a short-term development plan of the proposed approach and Backtester tool [1].

The next main milestone of the Backtester project is a development of flexible modular framework for development of mechanical trading systems. Such framework would incorporate user-defined modules that define metrics, indicators and mechanical trading algorithms, as well as learning-based components for the purpose of trading strategy optimization. Every aspect of the framework would be based on a strong formal basis, introducing mathematical rigor to a field of technical analysis and stock market trading.

The very nature of the technical analysis is that successful research results are highly confidential and proprietary. However, the field of methods and tools that support technical analysis and stock market trading has a perfect potential for further development. The Backtester software prototype [1] will be used as an integral component of open platform for development of mechanical trading systems.

References

1. "Backtester" homepage at SourceForge [HTML] (http://backtester.sourceforge.net).
2. Jeffrey Owen Katz, Donna L. McCormick. The Encyclopedia Of Trading Strategies. M.: Альпина, 2002. 388c.
3. Bruce James Vanstone, Clarence Tan A Survey of the Application of Soft Computing to Investment and Financial Trading. // Information Technology papers. Australia: Bond University, School of Information Technology, 2003. 6p.
4. Michael Korolyuk. How to compare trading strategies. Saint-Petersburg: Modern Trading, 2001. 11p.(In Russian)
5. Bruce James. Vanstone Trading in the Australian Stockmarket using Artificial Neural Networks. // Submitted to Bond University in fulfillment of the requirements for the degree Doctor of Philosophy. Australia: Bond University, School of Information Technology, 2005. 219p.
6. Sean D. Campbell A Review of Backtesting and Backtesting Procedures // Finance and Economics Discussion Series. Washington, D. C.: Federal Reserve Board, Divisions of Research & Statistics and Monetary Affairs, 2005. 23p.
7. Gautam Mitra, Triphonas Kyriakis, Cormac Lucas, Mehndi Pirbhai. A Review of Portfolio Planning: Models and Systems. Preprint. London, England: Brunel University, Center for the Analysis of Risk and Optimisation Modelling Applications, 2002. 43p.
8. Marcus Haas. New Methods in Backtesting. Preprint 2001-10. Bonn, Germany: Financial Engineering Research Center, 2001. 19p.
9. W. Hardle, G. Stahl Backtesting Beyond VaR. Preprint. Berlin, Germany: Institut fur Statistik und Okonometrie Wirtschaftswissenschaftliche Fakultat Humboldt-Universitat zu Berlin, 2000. 21p.
10. Colleen Cassidy, Marianne Gizycki Measuring Traded Market Risk: Value-at-Risk and Backtesting Techniques // Research Discussion Paper. Australia: Reserve Bank of Australia, Bank Supervision Department, 1997. 37p.
11. ARQA Technologies. Description of orders, available on the Russian stock market [HTML] (http://quik.ru/about/features/conditionalorders/). (In Russian)
12. Wikipedia.Org Feature comparison of Technical Analysis Software [HTML] (http://en.wikipedia.org/wiki/Technical_analysis_software#Feature_comparison_of_Technical_Analysis_Software).
13. Raphael Hertzog, Fabien Fulhaber. Genius Trader project homepage [HTML] (http://www.geniustrader.org).
14. pvoosten at users.sourceforge.net. LongBow project homepage [HTML] (http://sourceforge.net/projects/padamo/).
15. Andrew Leppardю Merchant of Venice project homepage [HTML] (http://mov.sourceforge.net).
16. Steve Stratos. Qtstalker project homepage [HTML] (http://qtstalker.sourceforge.net).
17. Glauco Siliprandi. QuantProject homepage [HTML] (http://sourceforge.net/projects/quantproject/).
18. Caleb Doise. Quote Blizzard project homepage [HTML] (http://sourceforge.net/projects/quoteblizzard/).