**NATIONAL RESEARCH UNIVERSITY HIGHER SCHOOL OF ECONOMICS**

**FACULTY OF ECONOMIC SCIENCES**

**BACHELOR’S PROGRAMME ECONOMICS**

**PROJECT PROPOSAL**

**MARKET MAKING EFFICIENCY EVALUATION WITH ML**

Valeriia Nizolia, BEC1710

Advisor:

Petr P. Lukyanchenko, Associate Professor

Department of Big Data and Information Search

Moscow

2021

This research is an attempt to answer whether machine learning methods can be used in market making problem. The study consists of two main parts: market making efficiency evaluation with classical machine learning algorithms and market making efficiency evaluation with reinforcement learning. Here is the second part of the research – RL algorithms. This part currently includes SARSA, Q-learning, DQN algorithms. The sample is represented by recent MOEX tick data (now algorithms are tested on Sberbank shares for several days in March 2019). Is it possible that the RL agent outperforms both the traditional market making strategies and strategies with ML in terms of cumulative P&L? According to the sources mentioned below, the results are supposed to be promising, however, there is no guarantee that they are before completing the practical part of the research.

**Introduction**

**Background.** Market making is one of the most crucial trading problems where the market liquidity is provided by an agent constantly buying or selling securities. There are some main issues in market making: the inventory risk, the risk of taking a bad position on market leading to losing significant amount of money (Spooner, 2018). Nowadays market making is becoming more automated because of huge dataflows and faster transactions` speed demanding instant agent’s reaction. It is impossible to name here all the algorithms studied to treat the problem of market making but reinforcement learning (RL) is worth mentioning. It has been used for many financial problems such as optimal execution and FX trading. RL also might be applied in a trading problem with a single trader whose actions cannot influence the market. RL algorithms converge to expected strategies in particular cases, however, challenges like order placement, cancellation and continuous state variables are not usually considered (Spooner, 2018).

**Problem Statement**. It can be noticed that the main research question in the majority of papers about market making with RL is whether RL agents can be applied to generate cumulative P&L but not whether they are able to outperform classical trading strategies used on real market. Besides, these papers do not contain the comparison of RL methods with classical ML methods used in algorithmic trading. The main *research question* of current study is whether strategies with RL can outperform other trading algorithms both traditional and using ML.

**Hypothesis.** RL methods used in market making are effective and competitive to other trading algorithms.

**Tasks and Delimitations of the Study.** To test the hypothesis, the following *tasks* should be completed:

1) to build and test different RL algorithms applied to market making problem;

2) to compare RL algorithms with traditional benchmarks and classical ML algorithms;

3) to interpret the results and to suggest new research questions.

**Professional Significance**. The professional significance of this study is the verification of whether RL algorithms can be applied to make profit by market makers on the real market.

**Literature Review**

*Market Making via Reinforcement Learning*

The authors design a market making agent with help of a self-simulated limit order book (LOB) (Spooner, 2018). They use temporal-difference RL, a linear combination of tile coding to approximate Q-learning function (three kinds of state representation) and original function for the cumulative reward instead of customary one (incremental P&L) to control the inventory risk (Spooner, 2018). The authors compare the performance of various TD-based learning algorithms (Spooner, 2018). RL agent must quote bid and ask prices at all time points while the inventory limits met. His actions are strongly restricted to a single buy/sell order without an opportunity to leave the market. The agent can choose to quote tight/wide, to skew in favour of any side or to clear its inventory. The P&L is computed through executions of the agent’s orders relative to the mid-price as for reward. It also included the inventory term showing the change in the agent’s cash. Besides the “natural” P&L reward function dampened reward functions are developed to capture the spread and to disincentivise trend-following behaviour. State representation consists of inventory, quoting distances normalised by spread, spread, mid-price move, queue imbalance, signed volume, volatility, relative strength index. These parameters are used for three state representations: agent-state and full-state (tile coding approximation scheme) and the approach where these two representations are approximated with help of an independent linear combination of their tile coding. Q-learning, SARSA, R-learning, Double Q-learning, Expected SARSA, On-policy R-learning and Double R-learning are tried to solve the problem. Among all algorithms SARSA might be considered the best one, and the most consistent (Spooner, 2018). As for reward functions, asymmetric dampening reward function with high dampening factor showed the highest risk-adjusted performance. The linear combination of tile coding was chosen for a better strategy divergence. The key point from this article is to pay attention to rewards and states which are the basis of the success of using RL in market making (Spooner, 2018).

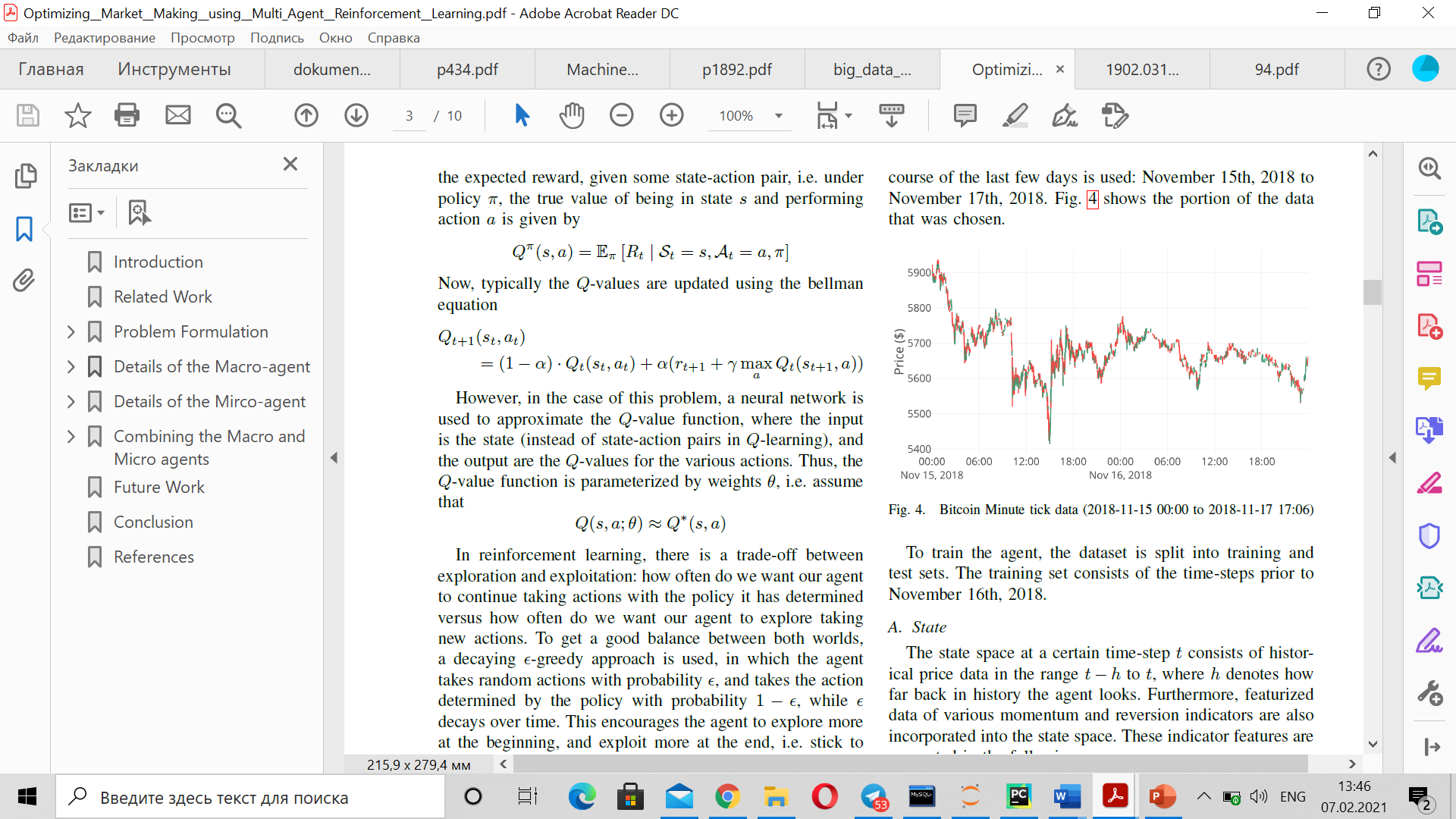
*Machine learning in Finance: From Theory to Practice, Chapter 9: Introduction to RL*

This book of Matthew F. Dixon et al. is worth mentioning because of a great number of cases providing an understanding of how RL works in optimization problems in finance (Dixon, 2020). High-frequency market making can be considered in the RL setting and results might be quite promising. An agent is learning to act optimally according to its main aim deciding which action to do in a certain environment coming from his previous actions and rewards. States are inputs, whereas agent’s actions are outputs, optimal behaviour mathematically means a maximization of objective outputs (Dixon, 2020). The main goal of market making RL agent is to capture the spread by placing limit orders to buy/sell. The inventory is restricted. The problem which market making agent must solve is when optimally to buy, sell or just hold. The price movement proxy is approximated by liquidity imbalance at the top of the LOB, according to this imbalance probabilities are filled (Dixon, 2020). The best bid/ask prices and volumes corresponding for them are provided by the market at each time update. The state representation is the product of inventory and a gridded liquidity ratio. When this ratio tends to zero, then mid-price is supposed to go up and ask is filled, the opposite situation is for liquidity ratio tending to one (Dixon, 2020). The expected P&L is chosen as a reward. The authors did not make great conclusions after the experiment except similar convergence of SARSA and Q-learning at the end (Dixon, 2020). This case demonstrated the basic RL implementation in market making, that is why it is important to mention it here.

*Machine learning for Trading*

The author is trying to answer if it is possible to train the RL algorithm to trade as a rational risk-averse investor (Ritter, 2017). Despite not including the certain problem of market making with RL, his research is significant in understanding why it is worth implementing RL to trading. RL is closely related to the Bellman theory of optimal control but unlike the latter, it is a way to directly learning value functions and solving optimal control problems. The key point of difference is that the value function is unknown and must be learned from the interactions between the environment and the agent. The author answers on the following important questions: how do abstract concepts of RL correspond with certain variables in trading strategy, how to consider the risk in RL algorithms and what are the mathematical assumptions on the random process on the market (Ritter, 2017). Firstly, the state space is defined to be all processes generating observations used by the agent to make the next trading decision. It usually contains the agent’s current position, prices, bid-ask spread and instrument’s liquidity. The current state can provide enough information for the agent to do the next action if the considered process is to be Markov (Ritter, 2017). The space of actions can be described as the space of trades or the space of target portfolios. However, the space of possible actions depends on the current state. The probability of taking the next action in a particular current state is called the agent`s policy and the agent searches for the policy which would maximize its expected total reward. In RL value functions are used to find the best policies but more information about them will be provided further. Q-learning was chosen as the main RL algorithm for the research (Ritter, 2017). As far as the reward is concerned, we need it to be a specific function of profit increments including the risk term. The expected wealth increment is represented by the changing value of agent`s portfolio. The cumulative reward where each reward is time-independent can be calculated as a simple sum of rewards. Asset returns are modelled with the mean-variance distribution. The key thoughts from this research are as follows: it is possible to make a risk-averse RL agent using Q-learning to find and then exploit the arbitrage opportunities (Ritter, 2017).

*Optimizing Market Making using Multi-Agent Reinforcement Learning*

This paper by Yagna Panel is perfect to understand how it is possible to use deep reinforcement learning to optimize market making (Patel, 2018). The author used a two-agent framework where each agent has its own role: macro-agent is used to make the decision to buy, sell or hold an asset, micro-agent is responsible for placing limit orders in the order book (Patel, 2018). This research is worth considering because it contains a good example of architecture of deep RL agent. Yagna Panel used a neural network to approximate the Q-value function with states as inputs and Q-values for the actions as outputs (Patel, 2018). That is why besides states and actions Q-value function using in a neural network contains weights *ϴ:*

The macro-agent is used to optimize decisions to buy, sell or hold corresponding to Q-value outputs of the neural network. The following parameters are used as state components – the inputs to a neural network: price level, price change, volume level, volume change, volatility and current assets list formulated each time the agent decides to buy, to sell or to hold. A deep Q-network is used to parametrize the value function: multilayer perceptron with two hidden layers with ReLu activation functions and Adam optimizer for training. After training even on its own the macro-agent shows a more profitable and stable strategy than simple strategies (Patel, 2018).

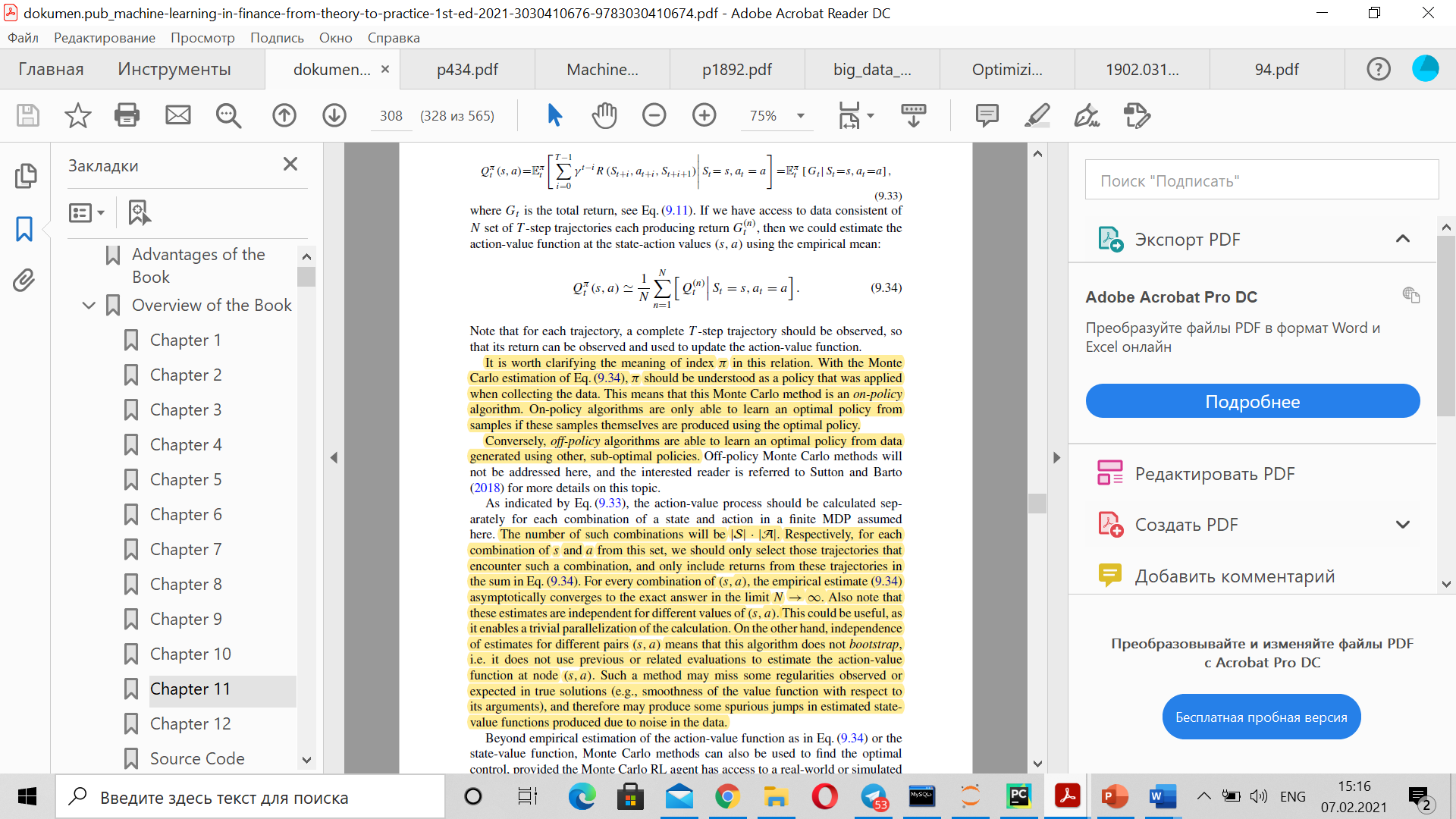
**Methods**

RL methods function similarly to dynamic programming methods in the optimization problems, however, there is difference in computational design and data processing (Dixon, 2020). There are three main classes of methods in RL: Monte Carlo methods, policy search methods and value-based RL.

*Monte Carlo methods*

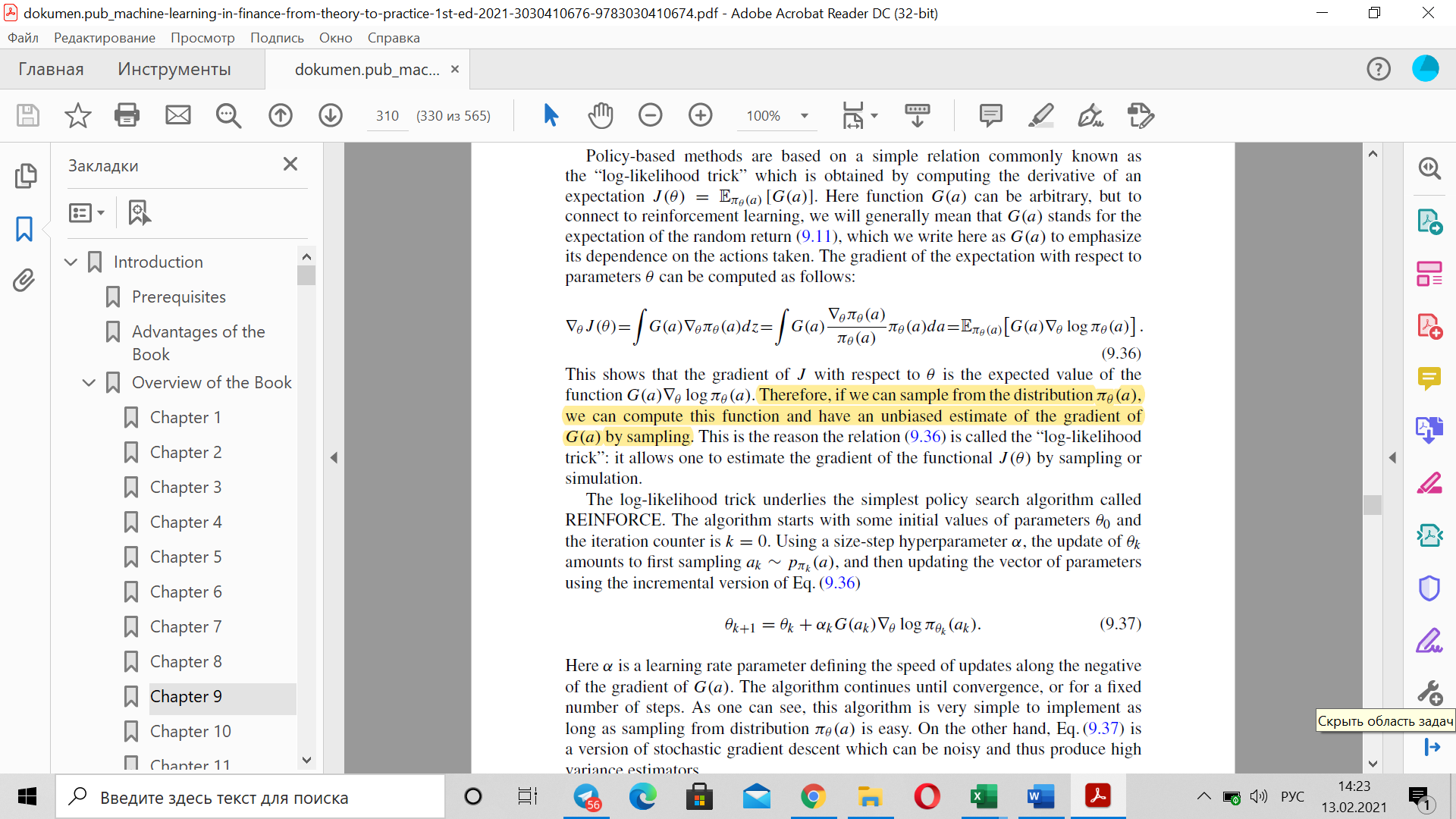
They rely on experience consisting of states, actions, and rewards. This experience can be even simulated to solve a certain optimization problem (Dixon, 2020). For Monte Carlo methods there is no need to have an explicit form of all transition probabilities, samples from them are enough. In the majority of cases Monte Carlo methods assume the finite time horizon and work according to action-value function:

where Gt – total reward, s – state, a – action.

It is possible to approximate action-value function at the state-action values with mean if there is enough consistent data of N set of T-step trajectories (Dixon, 2020):

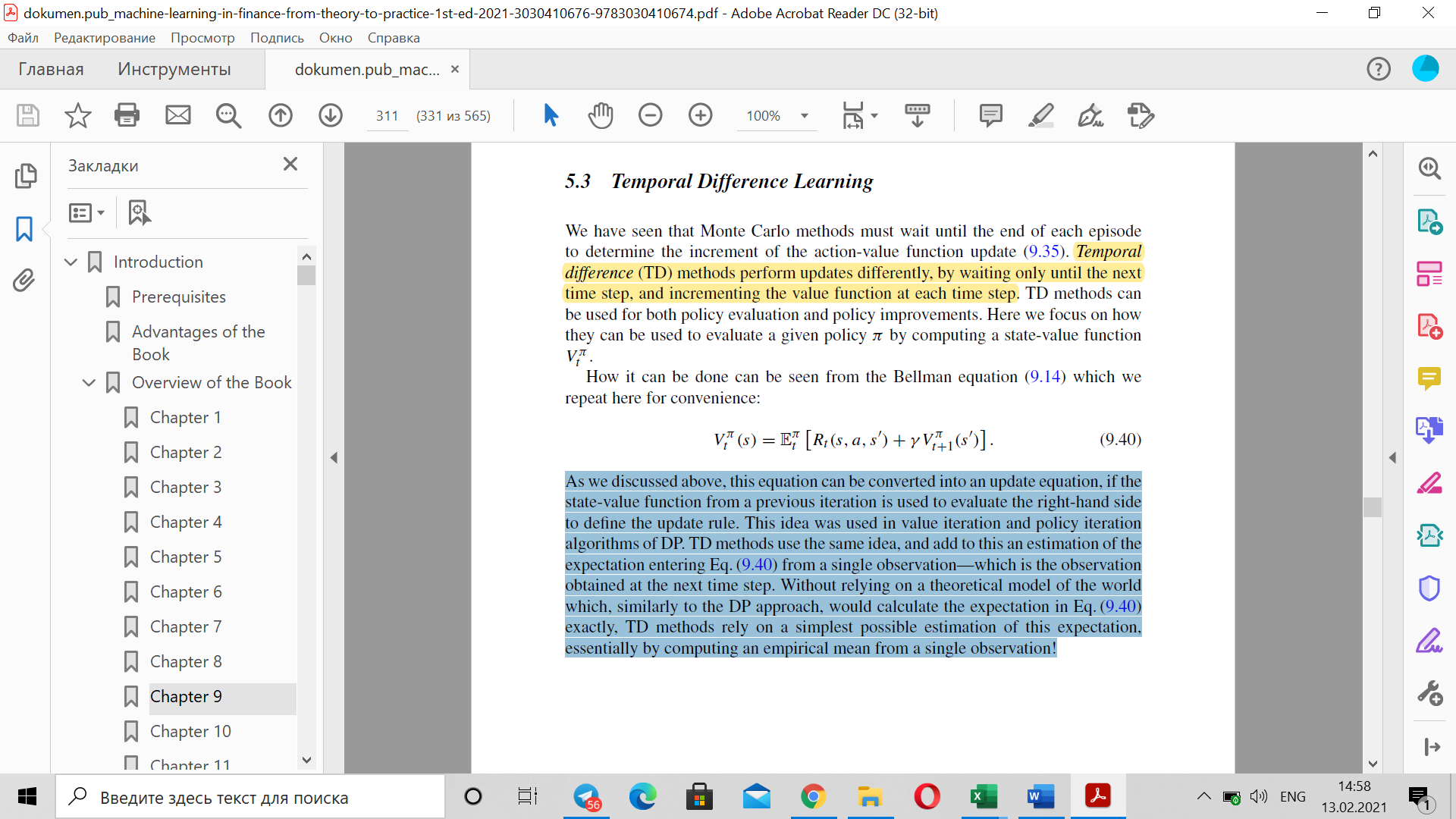
The action-value process is calculated separately meaning that the number of combinations (s, a) equals to |S|\*|A| encountering a combination and including returns from trajectories in the sum for empirical mean written above. Each combination (s, a) corresponds to the mean asymptotically converging to the exact answer tending to infinity (Dixon, 2020).

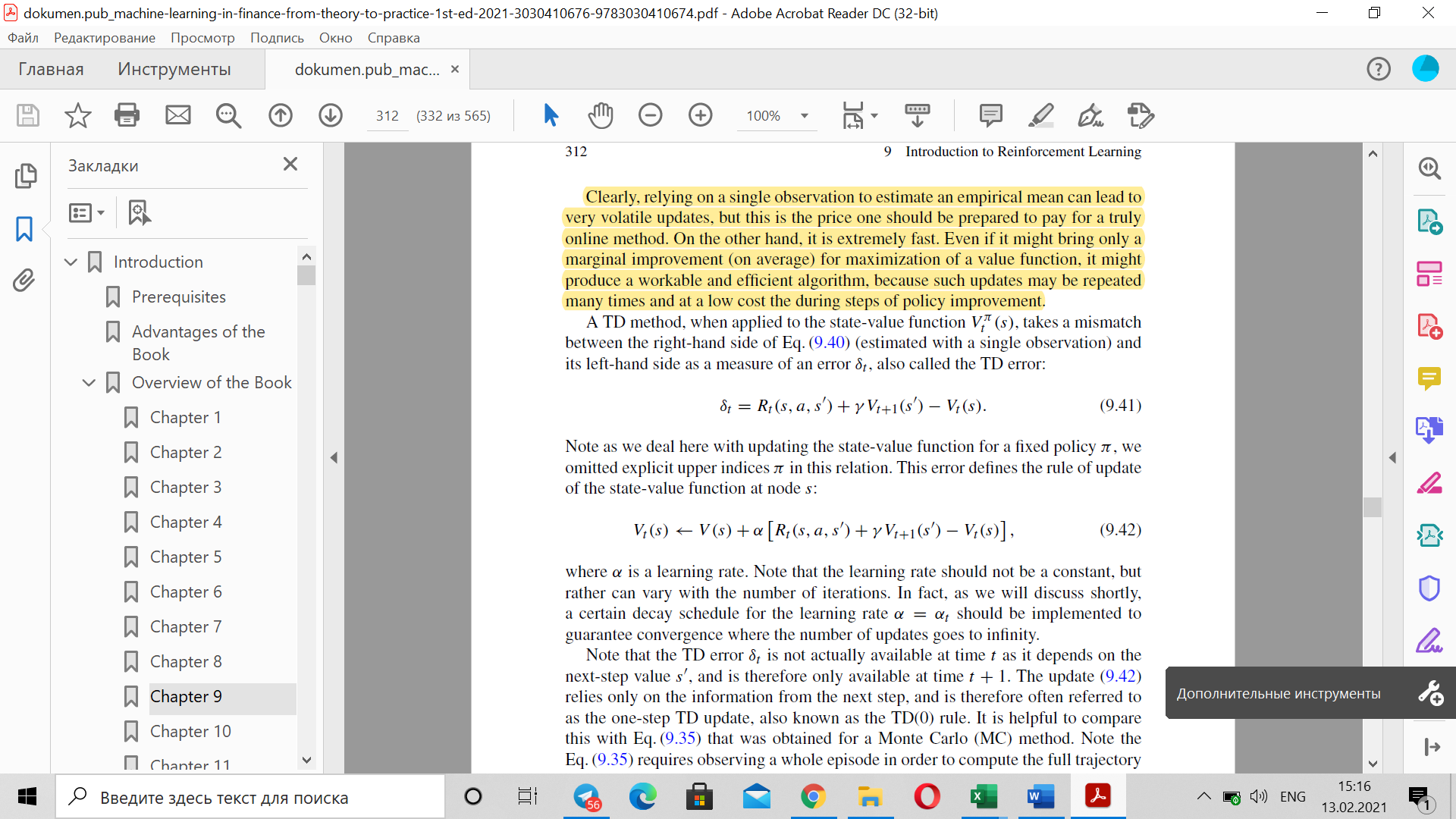
*Policy-based learning*

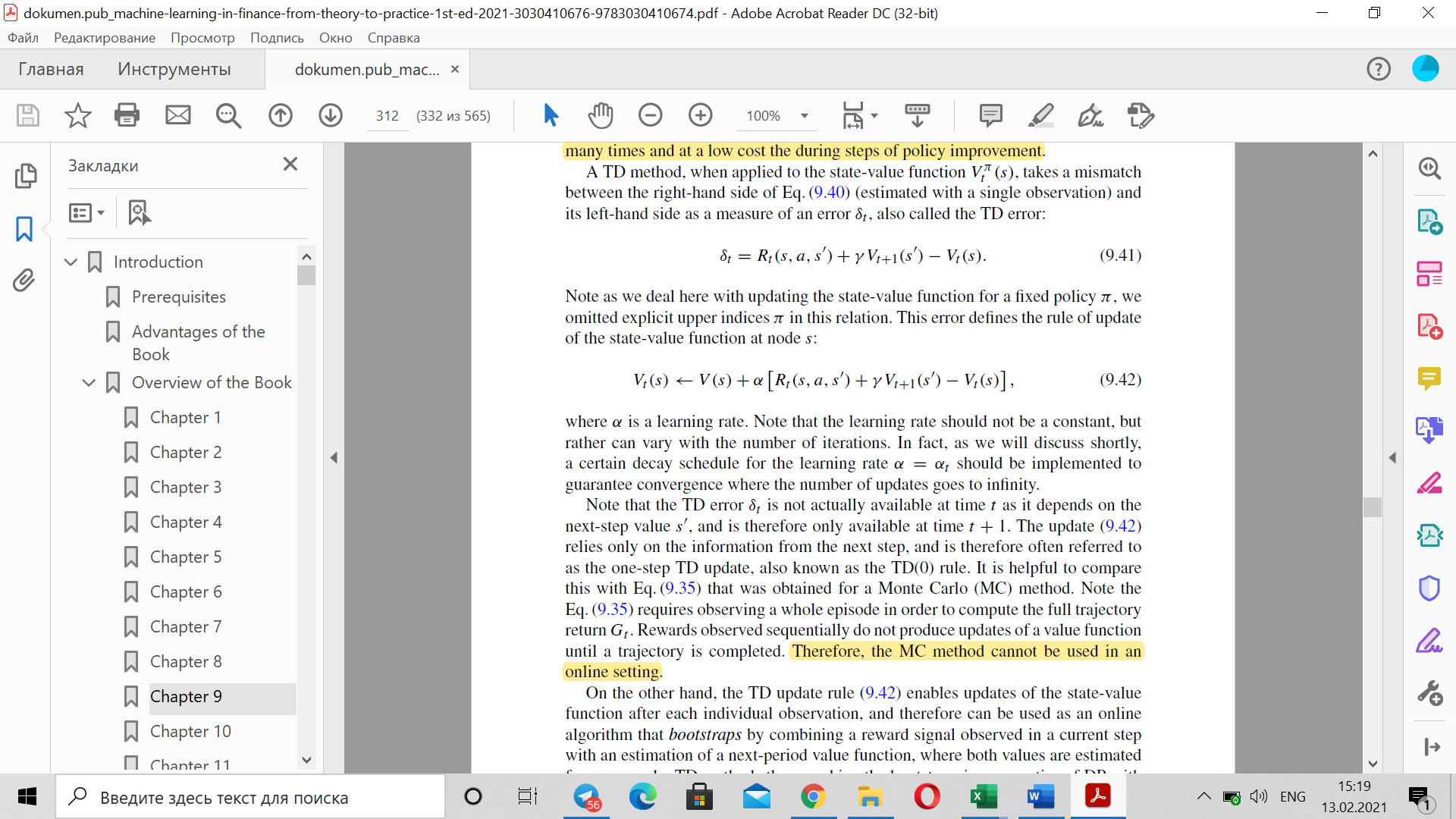
Policy-based learning models policy directly. Stochastic policies *πθ (a|s)* define probability distributions over actions *a ∈ A* with *θ* defining distribution parameters. The main aim of learning is to maximize the cumulative expected reward by tuning parameters (Dixon, 2020). Policy-based methods are based on the following relation the key part of which is derived by computing the derivative of the expectation of the random return:

This policy search algorithm is called REINFORCE (Dixon, 2020). It starts from some initial values *θ0* where k (iteration counter) equals to 0 and then continues updating the parameters according to the formula written above with the speed α until convergence or for the defined number of steps.

*Value-based methods*

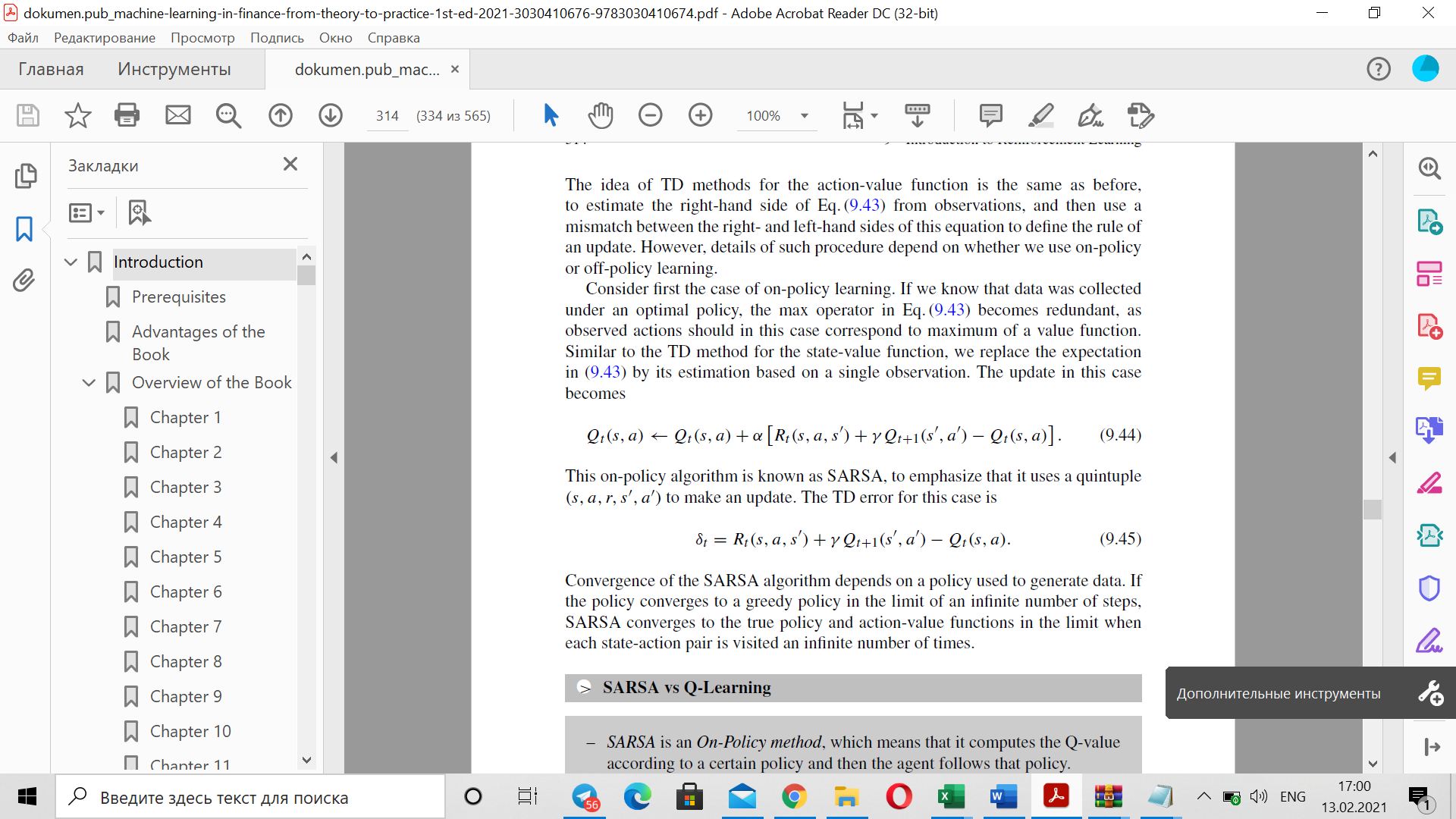
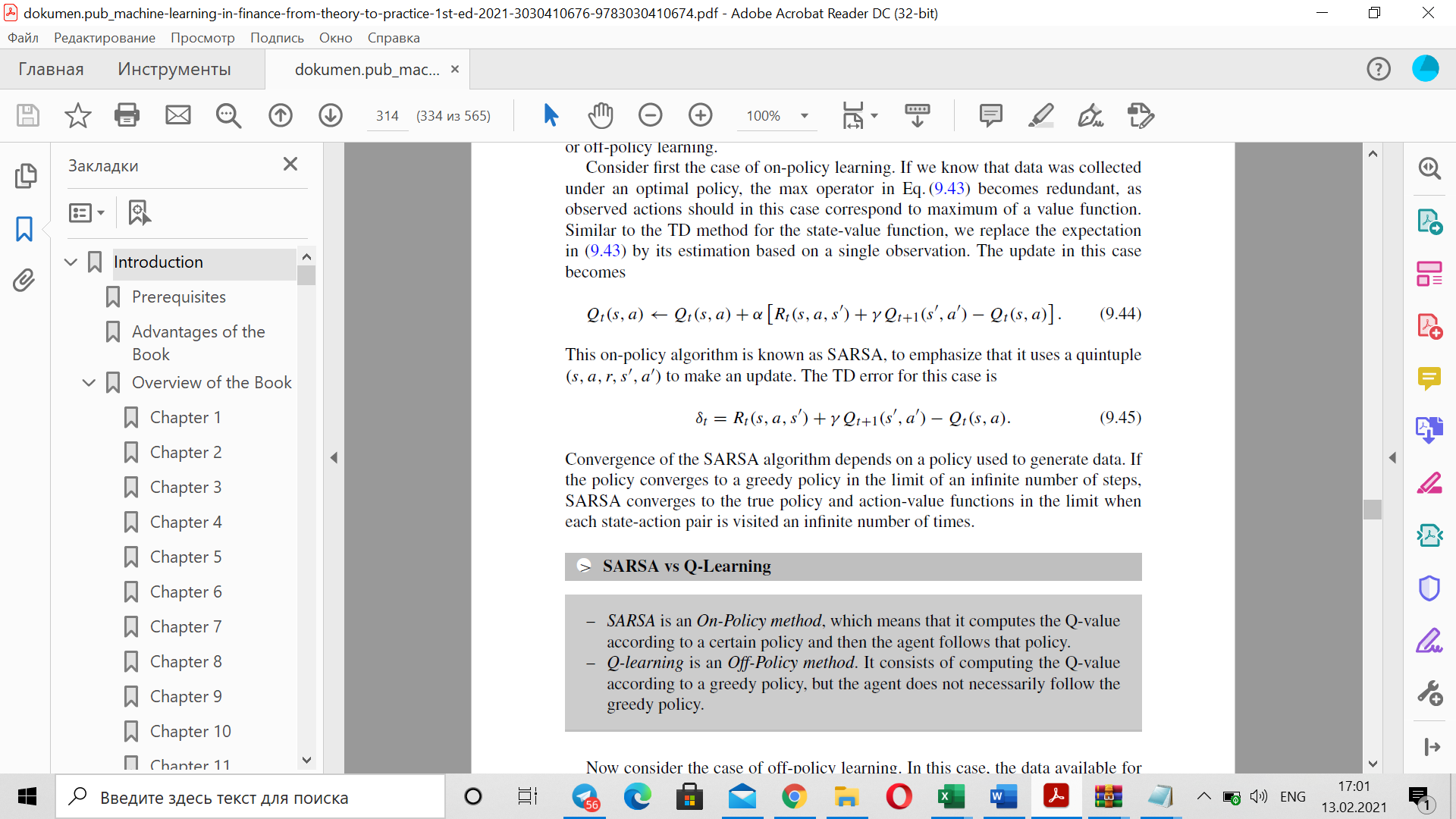
Before the explanation of functioning of value-based algorithms, it is necessary to define temporal difference methods performing updates by waiting until the following step. They can be used in policy evaluation by computing a state-value function. TD methods repeat the idea of the Bellman equation:

and add an estimation of the expectation entering the mentioned equation from a single next observation obtained using an empirical mean (Dixon, 2020). Applying to the state-value function the TD method takes an error between the right- and left-hand sides of the Bellman equation estimated with a single observation (Dixon, 2020):

The error is used to update the state-value function (Dixon, 2020):

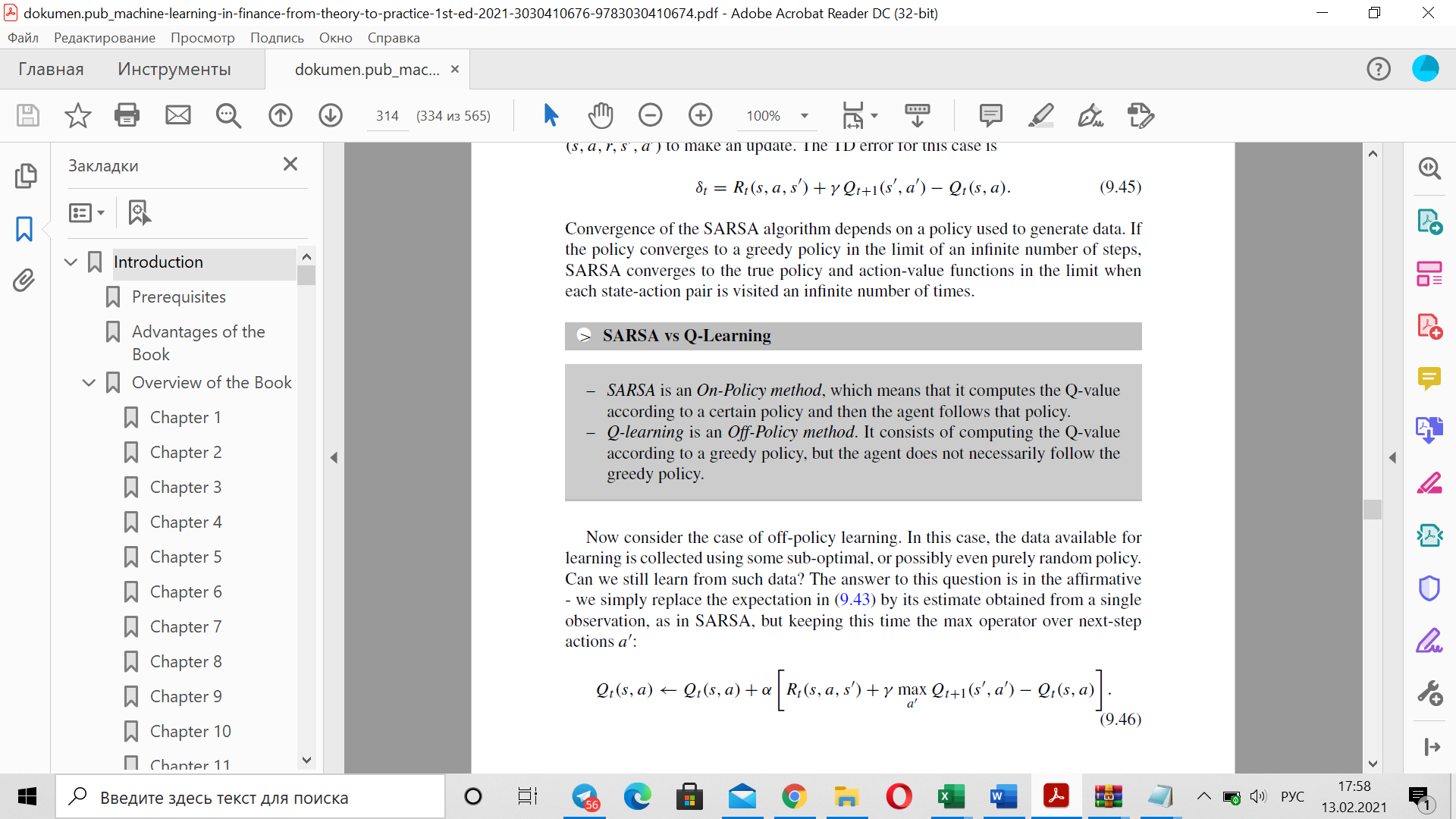
where *α* is a learning rate changing with the number of iterations. TD methods are helpful in practice due to their updates after each observation. Moreover, they can be applied not only to state-value function but to action-value function too and in both cases multi-step updates TD (*λ*) can be used (Dixon, 2020).

*SARSA*

One of the most famous RL methods based on action-value function *Q (s, a)* is SARSA. It is an “on-policy” algorithm assuming that the optimal policy function is already known, and it must be learned from the data (Sutton&Barto, 2018). If we know the optimal policy, then observed actions correspond to the maximum of a value function and the update and TD error look the following way:

Convergence of the SARSA method is directly dependent on the way to generate data (Dixon, 2020).

*Q-learning*

An off-policy analog of the SARSA algorithm but even more popular is the Q-learning algorithm which can learn from the data collected with help of sub-optimal or random policy (Sutton&Barto, 2018). In this case, we just need to keep the max operator over the next-step actions according to the following policy:

Q-learning is a TD (0) learning which is applied to an action-value function. As for the convergence, the Q-learning algorithm is supposed to converge for finite MDPs with learning rate’s decay. Q-learning is convenient for many practical tasks, for example, it used in batch RL where the agent learns from the data previously obtained from another agent (Sutton&Barto, 2018).

*Deep reinforcement learning*

Deep reinforcement learning is mainly based on neural networks to approximate value and policy functions (Dixon, 2020). For instance, *Q (s, a)* can be considered as a multilayer neural network where parameters ϴk are weights of a last linear layer of the network. The update rule now looks the following way:

where *w* and *w-* are vectors ofparameters to compute *q (s, a)* and *q (s’, a’)* (target network) accordingly. When computing the update, gradients with respect to *w-* must not be considered because they are fixed (Dixon, 2020). With new transitions old transitions are deleted and parameters update with help of a mini-batch from the buffer with stochastic gradient descent update. Sometimes agent chooses not optimal action but a random one to explore the environment (Dixon, 2020).

**Results Anticipated**

To begin with, a simple problem setting has been used. Only one price level was taken for RL agent to make his choice of further action to capture the bid-ask spread. Also, the inventory was strictly limited to [-1;1], the liquidity ratio was used as a proxy for price movement to quote bid or ask or just stay stable till the next update. SARSA and Q-learning were used to learn optimal strategy in such problem setting and only cumulative P&L was calculated to show that RL algorithms can work quite well.

However, the cash the RL agent had was not limited what is impossible on real market. The next action of the current research is to set the limit on cash which highly likely leads to increasing the costs and the time of keeping the position. Moreover, it is necessary to define agent’s costs, experiment with different inventory limits, to include limit order placement at different levels of the ladder.

Furthermore, other RL algorithms must be used to solve the market making problem, for example, it is challenging to apply neural networks to approximate the Q-learning function.

Finally, a comparison with traditional trading strategies and strategies including ML must be made. According to current research, results about RL agent’s ability to outperform other trading algorithms are expected to be positive.

**Conclusion**

The main anticipated *result* of this research is that RL algorithms can be considered as decent competitors to traditional trading algorithms. However, even now it is possible to name the key problems of RL like partial observability, the challenges to define the environment and the slower speed of these algorithms due to their difficulty. The final answer about RL in market making is expected to be given in the current research.

Word Count 2500

# References

Dixon, Matthew F., Halperin, Igor, Bilokon, Paul. (2020). *Machine learning in finance: from theory to practice.* Cham, Switzerland: Springer Nature Switzerland AG

Richard S. Sutton and Andrew G. Barto. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Ritter, Gordon. (August 8, 2017). *Machine Learning for Trading*. Available at SSRN: https://ssrn.com/abstract=3015609 or <http://dx.doi.org/10.2139/ssrn.3015609>

Spooner, Thomas & Savani, Rahul & Fearnley, John & Koukorinis, Andreas. (2018). *Market Making via Reinforcement Learning*. In Proc. Of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Stockholm, Sweden

Théate, T., & Ernst, D. (2020). *An Application of Deep Reinforcement Learning to Algorithmic Trading*. ArXiv, abs/2004.06627.

Pankaj Kumar. (2020). *Deep Reinforcement Learning for Market Making*. In Proc. Of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020, IFAAMAS, 3 pages.

Patel, Y. (2018). *Optimizing Market Making using Multi-Agent Reinforcement Learning*. ArXiv, abs/1812.10252.