ISCTE Business School
Instituto Universitário de Lisboa

Adaptive Value-at-Risk policy optimization: a deep reinforcement learning approach for minimizing the capital charge

Guilherme Sousa Falcão Duarte Banhudo

Dissertation submitted as partial requirement for the conferral of
Master in Finance

Supervisor:
Dr. António Barbosa, Assistant Professor, ISCTE Business School, Finance Department

October, 2019

Abstract

In 1995, the Basel Committee on Banking Supervision emitted an amendment to the first Basel Accord, allowing financial institutions to develop internal risk models, based on the value-at-risk (VaR), as opposed to using the regulator's predefined model. From that point onwards, the scientific community has focused its efforts on improving the accuracy of the VaR models to reduce the capital requirements stipulated by the regulatory framework. In contrast, some authors proposed that the key towards disclosure optimization would not lie in improving the existing models, but in manipulating the estimated value. The most recent progress in this field employed dynamic programming (DP), based on Markov decision processes (MDPs), to create a daily report policy. However, the use of dynamic programming carries heavy costs for the solution; not only does the algorithm require an explicit transition probability matrix, the high computational storage requirements and inability to operate in continuous MDPs demand simplifying the problem. The purpose of this work is to introduce deep reinforcement learning as an alternative to solving problems characterized by a complex or continuous MDP. To this end, the author benchmarks the DP generated policy with one generated via proximal policy optimization. In conclusion, and despite the small number of employed learning iterations, the algorithm showcased a strong convergence with the optimal policy, allowing for the methodology to be used on the unrestricted problem, without incurring in simplifications such as action and state discretization.

JEL Classification Numbers: G21, G28, C45

Keywords: Value at Risk, Basel Accords, Artificial Intelligence, Deep Learning, Deep Reinforcement Learning, Proximal Policy Optimization

Resumo

Em 1995 foi emitida uma adenda ao Acordo de Basileia vigente, o Basileia I, que permitiu que as instituições financeiras optassem por desenvolver modelos internos de medição de risco, tendo por base o value-at-risk (VaR), ao invés de recorrer ao modelo estipulado pelo regulador. Desde então, a comunidade científica focou os seus esforços na melhoria da precisão dos modelos de VaR procurando assim reduzir os requisitos de capital definidos na regulamentação. No entanto, alguns autores propuseram que a chave para a optimização do reporte não estaria na melhoria dos modelos existentes, mas na manipulação do valor estimado. O progresso mais recente recorreu ao uso de programação dinâmica (DP), baseada em processos de decisão de Markov (MDP) para atingir este fim, criando uma regra de reporte diária. No entanto, o uso de DP acarreta custos para a solução, uma vez que por um lado, o algoritmo requer uma matriz de probabilidades de transição definida, e por outro, os elevados requisitos de armazenamento computacional e incapacidade de lidar com processos de decisão de Markov (MDP) contínuos, exigem a simplificação do problema em questão. Este trabalho visa introduzir *deep reinforcement learning* como uma alternativa a problemas caracterizados por um MDP contínuo ou complexo. Para o efeito, é realizado um *benchmarking* com a *policy* criada por programação dinâmica, recorrendo ao algoritmo *proximal policy optimization*. Em suma, e apesar do reduzido montante de iterações empregue, o algoritmo demonstrou fortes capacidades de convergência com a solução óptima, podendo ser empregue na estimativa do problema sem incorrer em simplificações.

Acknowledgement

I would like to thank all of those involved in the process of creating this thesis for their support and assistance. From my supervisor Professor António Manuel Barbosa; to my family and friends.

Index

1 Introduction 1
2 Theoretical Framework 4
2.1 Basel Accords 4
2.2 Value at Risk 7
2.3 Markov Decision Process 11
2.4 Proximal Policy Optimization 13
3 Literature Review 16
4 Methodology 19
4.1 Environment Model 19
4.2 Results 25
4.2.1 Computational Considerations 26
4.2.2 Training Progress 28
4.2.3 Policy Analysis 30
4.2.4 Policy Benchmarking 39
4.2.5 Model Limitations 47
4.2.6 Policy Gradient Advantages 47
4.2.7 Deep Reinforcement Learning Limitations 49
4.2.8 Sub-optimal Optimality 50
5 Conclusion 51
6 Future Work 53
7 Bibliography 56
8 Appendixes 58
Appendix A – Theoretical Foundations on Deep Reinforcement Learning 58
Appendix A.I – From Dynamic Programming to Reinforcement Learning 58
Appendix A.II – Deep Reinforcement Learning 64
Appendix A.III – Policy Gradient 66
Appendix B – The reinforcement learning policy's plot as a function of the time remaining until the backtesting process resumes 71
Appendix C - Source Code 76

1 Introduction

In 1995, the Basel Committee on Banking Supervision issued an amendment to the first Basel Accord, Basel I. This amendment allowed financial institutions to develop internal risk models, based on the value-at-risk (VaR), as opposed to using the regulator's predefined model. However, this liberty came at a cost, as the model's ability to capture observed risk would be assessed on a yearly backtesting process, penalizing the market risk charge (MRC) as a function of the recorded violations. From this point onwards, the financial community focused its efforts on improving the accuracy of the VaR models to reduce the regulatory capital requirements. However, some authors proposed that the key towards disclosure optimization would not lie in improving the existing models, but in manipulating the estimated value. The most recent progress in this field employed dynamic programming (DP) based on Markov decision processes (MDPs), to create a daily report policy, based on the applicable regulation and the model's ability to capture observed risk. An issue with the use of dynamic programming is the heavy cost involved. Firstly, the algorithm requires an explicit transition probability matrix, unfeasible in financial markets. This, paired with high computational storage requirements and an inability to operate in continuous MDPs, meant an alternative solution was needed.

In recent years, neural networks (NNs) have seen a widespread growth, boosting the fields which rely on its use, from computer vision to data science. However, the usage of these techniques within the financial sector has been long overdue, partly, due to the concerns over the lack of transparency of black-box methodologies. The purpose of this work is to introduce deep reinforcement learning (DRL) as an alternative to solving problems characterized by MDPs, whose dynamics have proven to be too complex or hard to map, usually the case in financial markets. The introduced class of algorithms do not require an explicit transition matrix, and therefore require little information about the underlying environment's dynamics. Furthermore, some versions have the ability to learn continuous action and state spaces. In short, the goal of reinforcement learning is to provide an optimal policy which maps states to actions through a repeated trial-and-error process.

In order for the algorithm to be deemed viable to tackle this category of financial problems, its ability to converge towards a known optimal solution must be assessed, in other words, benchmarked, under the premise that should the algorithm prove capable of approximating a simple problem's solution, then surely its capability will be maintained when solving a complex environment, in which dynamic programming is not viable. For this purpose, proximal policy optimization (PPO), one of the most recent developments in DRL with improved convergence and stability in the continuous domain in comparison to its predecessors, has been selected to approximate the optimal solution computed by DP. In practice, the usage of algorithms with solid performance in discrete spaces, such as double dueling deep Q-network (DDQN), would be more appropriate for the selected benchmark as they can estimate the optimal solution faster and more efficiently – the algorithms in the policy gradient taxonomy, to which PPO belongs, tend to be stuck in local optima.

The problem selected for benchmarking is that of optimizing the value at risk disclosure under the second Basel Accord, characterized by discrete action and state spaces. The reasoning behind this selection is due to the fact that the problem in question has a complex and demanding environment, regarding its space dimensions, which could benefit from proximal policy optimization's ability of learning under continuous spaces. This would therefore avoid many of the pitfalls and simplifications incurred to make the problem tractable for dynamic programming. The solution consists in the creation of a policy, that is, a map from actions to spaces, in which the state corresponds to a tuple of (a) the time remaining until the backtesting process is resumed, in days, in which the multiplier is reviewed as a function of the incurred exceedances, (b) in exceedances recorded to date, and lastly, (c) the applicable multiplier, for the relevant period. The policy is constructed via a proximal policy optimization agent, which learns the dichotomy between reporting low VaR values, minimizing the short-run cost, and the occurrence of exceedances, in which the observed loss surpasses the reported expected loss, leading to the next year's multiplier to be modified.

The first part of this thesis introduces the applicable theoretical base, from the regulatory context which originates the need to optimize an internal model's VaR disclosure, to the deep reinforcement learning theory, seeking to establish a link between dynamic programming and reinforcement learning. The focus then shifts to approximating the optimal policy generated by Seixas (2016), in addition to benchmarking the yielded solution with that of the mentioned author's. Despite the small number of training iterations used in approximating the solution, the algorithm's policy presented strong signs of convergence with the DP's optimal policy, yielding similar results in behavior and incremental return, generated through the investment of the freed capital, whilst maintaining the institution's exposure constant.

The thesis’ contribution to the field is therefore threefold: (a) it demonstrates the adequacy of deep learning in providing a solution to the Basel disclosure problem; (b) it paves the way for future improvements on the problem at hands via deep reinforcement learning; and lastly, (c) it introduces the class of deep reinforcement learning as a solution for optimization problems, a methodology long overlooked in the financial community.

2 Theoretical Framework

2.1 Basel Accords

In late 1974, in the wake of severe disturbances in the currency and banking markets, the G10 central bank governors established the Committee on Banking Regulations and Supervisory Practices. This committee was posteriorly renamed to Banking Committee on Banking Supervision (BCBS).

The BCBS was created as a regulatory body, providing a framework for global supervision and risk regulation. Nevertheless, the BCBS does not have legal power, nor has it pursued such goal. The institution's purpose is, in short, to provide guidelines and standards on banking regulation, and to create a channel for cooperation and discussion between financial institutions.

In July 1988, the Basel Committee on Banking Supervision presented a framework for measuring the capital adequacy, specifically, to establish the minimum levels of capital required for international banks. This documentation became known as Basel I, suggesting there would be further improvements to the document. The first Basel Accord focused mainly in credit risk, coupling the required capital level with the degree of credit risk in the institution's portfolio. The latter's assets would then be categorized into three buckets according to its risk, as perceived by the regulator. The regulation stipulated that financial institutions (FI) were required to hold a minimum of eight percent of capital in relation to risk-weighted assets – the capital adequacy ratio (CAR). Formally, this notion is expressed by the following formula:

$$CAR = \frac{Tier\ I\ Capital + Tier\ II\ Capital}{Risk\ weighed\ assets\ for\ credit\ risk} 100\,, \tag{1}$$

where $Tier\ I\ Capital$, often referred to as core capital, comprises (a) paid up capital, (b) reserves and surplus, and (c) capital reserves and $Tier\ II\ Capital$, termed supplementary capital, refers to (a) undisclosed reserves, (b) revaluation reserves, (c) general provision and loss reserves, (d) hybrid (debt/equity) capital instruments, and € subordinated debt instruments. Additionally, the accord provided means of separating an institution's assets into five different percentage categories, based on its risk nature, and accordingly, establishing each asset's weight-factor.

Following general criticism that the standard approach defined in Basel I was incapable of accurately measuring risk, the BCBS incorporated market risk in capital requirements, reflecting the increasing tendency for FIs to increase their exposure to derivatives. In addition, the document introduced the ability for firms to self-regulate, as a means of encouraging risk taking and adequate measurement. The 1995 amendment to the first Basel Accord provided a framework for firms to assess the quality of their risk models through a backtesting process, as this methodology started to diffuse among FIs. Following this document, institutions were allowed to develop their own financial models to compute their market risk capital thresholds – the daily VaR. This archetype would be known as the Internal Model Approach (henceforth referred to as IMA). The backtesting process tied the market risk capital requirement, MRC, to both the portfolio's risk, and the internal model's quality. The market risk charge in a given day t, would then correspond to the combination of two components, the general risk charge and the specific risk charge

$$MRC = GRC + SRC\,, \tag{2}$$

where GRC and SRC correspond to the general and specific risk charges, respectively. Whereas the GRC depended directly on the model, SRC represented the specific risk charge, a buffer against idiosyncratic factors, including basis and event risks. The former corresponded to the maximum between the present day's value-at-risk, and the average of the last sixty daily risk disclosures, multiplied by a factor, which became known as k or the multiplier factor. The condition stated in the previous sentence reflects in the following mathematical expression

$$GRC_t = \max\left(VaR_{t,1\%}, k * \frac{1}{60}\sum_{i=0}^{59} VaR_{t-i,1\%}\right), \tag{3}$$

where the multiplier's value, k, depended on the amount of violations verified during the backtesting process, and $VaR_{t,1\%}$ represents the 10-day value-at-risk computed at the 1% level.

Zone	Number of Exceptions	Potential Increase in K	Multiplier Value (K)	Cumulative Probability (%)[1]
Green	0 to 4	0.00	3.00	[8.11;89.22]
Yellow	5	0.40	3.40	95.88
	6	0.50	3.50	98.63
	7	0.65	3.65	99.60
	8	0.75	3.75	99.89
	9	0.85	3.85	99.97
Red	≥10	1.00	4.00	99.97

Table 1 – The Basel Penalty Zones

The backtesting process would take place every 250 trading days, analyzing the entire period's model estimates, with the VaR in question being computed at the 1% significance level. Table 1 summarizes the multiplier factor k's states in relation to the recorded exceedances in the backtesting process for a given period.

In 2004, the BCBS released Basel II, the second Basel Accord. The document sought to further improve the risk management and capital adequacy guidelines set by Basel I, sixteen years earlier. The motivation behind this improvement lied in Basel I's innability to differentiate risk, especially among members of the Organization for Economic Cooperation and Development (OECD), and the discrepancy between Basel I's risk weights, and the actual economic risks. Whilst the first accord focused on credit risk, the new proposal integrated market (included in the 1996 ammendment) and operational risks on the minimum capital requirement computation. Another important addition in Basel II was the fact that assets' credit rating played a major role in determining risk weights. Such reflected in riskier assets having larger weights, thus leading to a larger MRC.

Basel II created a three pillar structure, (a) minimum capital requirements, (b) supervisory review process and lastly, (c) market discipline. Regarding the first pillar (a), its goal was to provide a framework for calculating the required capital level for specific risk types, namely credit, operational and market risks. The first branch provided FIs with several alternatives for computing each of the risk types, thus enabling firms to choose that which suits their risk

[1] "The probability of obtaining a given number or fewer exceptions in a sample of 250 observations when the true coverage level is 99%" (Basel Committee on Banking Supervision, 1996).

profile, enabing exceptional loss or economic crysis endurance. Pillar number two (b) provided details regarding how supervision should be organized in order to ensure the implementation quality of internal processes and controls, resulting in additional capital levels when applicable. Said pillar relied on the use of banking stress tests to assess an institution's strength in adverse economic scenarios. Lastly, the third pillar (c) concerns transparency, referring to mandatory disclosures within each FI to the general public, thus enabling symmetric market information, whilst facilitating FI comparison.

At the time of the present work, the fourth Basel Accord was already showing signs of replacing its predecessor. However, since the content reflected throughout the document focuses on the second Basel Accord, the succeeding framework has not been further discussed.

2.2 Value at Risk

The value-at-risk is a statistical measure of potential loss, currently the market's standard measure in assessing market risk. This concept can be defined as the maximum potential change in a financial portfolio's value, with a certain probability, over an established period of time (Alexander, 2008).

According to Artzner et al. (1999), a risk measure $\rho(\cdot)$ is to be considered coherent should it abide by four principles: (a) the monotonicity condition, which states that if a portfolio has lower returns than another portfolio for every state of the world, the latter's risk measure should be greater than the former's; (b) the translation invariance property, which establishes that if an amount of cash is added to a portfolio, its risk measure should go down by the same amount; (c) the homogeneity requirement in turn, stated that changing the size of a portfolio by a factor λ, while keeping the relative amounts of different items in the portfolio the same, should result in the risk measure being multiplied by λ; and lastly, (d) the subadditivity axiom states that the risk measure for two portfolios after they have been merged, should be no greater than the sum of their risk measures before said transformation. These conditions are represented mathematically in table 2.

Condition Name	Condition Expression
Monotonicity	$\rho(Y) \geq \rho(X)\ if\ X \leq Y$
Homogeneity	$\rho(\alpha X) = \alpha\rho(X), \forall\ \alpha > 0$
Risk Free Condition	$\rho(X + k) = \rho(X) - k, \forall\ k$
Subadditivity	$\rho(X + Y) \leq \rho(X) + \rho(Y)$

Table 2 – The four coherent risk measure conditions

Value-at-risk satisfies the first three conditions, but is not guaranteed to satisfy the fourth condition, the subadditivity axiom.

The value at risk can be described as the maximum loss which can be expected to occur if a portfolio is held static for a given amount of time h, under a certain confidence level $(1 - \alpha)$ (Alexander, 2008). In a more practical view, it can be thought of as the amount of capital that must be added to a position to make its risk acceptable to regulators. This risk measure was introduced as an alternative to standard portfolio risk metrics, namely volatility and correlation, as these can only accurately measure risk when the asset's or risk factor's returns have a multivariate normal distribution. Value-at-risk encompasses a wide set of attractive features, namely the fact that it can easily be aggregated and disaggregated whilst taking into consideration the dependencies between its constituents; and its ability to not only measure the risk factor's risk, but their sensitivities as well (Alexander, 2008).

Considering a significance level, α, such that, $0 < \alpha < 1$, its quantile for a given distribution is given by

$$P(X < x_\alpha) = \alpha\ , \tag{4}$$

Thus, the quantile α of distribution X, x_α, can be obtained according to

$$x_\alpha = F^{-1}(\alpha)\ , \tag{5}$$

where F^{-1} represents the inverse of the distribution function.

Recall the VaR corresponds to the maximum loss which is expected to be exceeded with probability α, when the portfolio is held static for h days. Accordingly, it corresponds to the computation of the α quantile of the discounted h-day P&L distribution:

$$P\left(X_h < x_{ht,\alpha}\right) = \alpha\ , \tag{6}$$

where α corresponds to the significance level and $X_h = \frac{B_{ht}P_{t+h}-P_t}{P_t}$ represents the h-day discounted return.

According to the previous statement, and since the value-at-risk is an estimated loss, its value can be obtained by direct application of equation (5):

$$VaR_{h,\alpha} = -F_L^{-1}(\alpha)\ , \tag{7}$$

where $F_L^{-1}(\alpha)$ represents the inverse cumulative distribution function of losses.

By replacing the previous equation into (6) yields

$$VaR_{h,\alpha} = -x_{ht,\alpha}\ , \tag{8}$$

According to Simons (2000), the parametric linear framework is the most used of the three existing estimation methods, hence, this thesis' focus. Within parametric methods, most research focuses on the use of normal distribution given its simplicity and the ability to use the h-day square root, $\sqrt{h}$, as a scaling rule for linear portfolios. It is important to note however, that this rule leads to a systematic underestimation of risk, where the degree of underestimation is aggravated the longer the time horizon, jump intensity[2] and confidence level, failing to address the objective of the Basel Accords (Danielsson & Zigrand, 2003). Nevertheless, this thesis will focus on the normal parametric value-at-risk and its scaling rule, given its well-known behavior and widespread use.

Assuming the portfolio's discounted returns, $X_{h,t}$ are i.d.d. and normally distributed with mean μ and standard deviation σ, i.e.

$$X_{t,h} \overset{idd}{\sim} N(\mu_{ht}, \sigma_{ht}^2)\ , \tag{9}$$

applying the normal standard transformation to the previous variable yields

$$P\left(X_{ht} < x_{ht,\alpha}\right) = P\left(Z < \frac{x_{ht,\alpha} - \mu_{ht}}{\sigma_{ht}}\right) = \alpha\ , \tag{10}$$

where Z is a standard normal variable.

[2] In The term jump refers to jump diffusion stochastic processes, models which aim at assessing the probability of two i.d.d. variables modeled under the same distribution, seeing their price move significantly and synchronously.

Applying equation (5) to the previous expression results in the portfolio's standardized discounted return's inverse cumulative function

$$\frac{x_{ht,\alpha} - \mu_{ht}}{\sigma_{ht}} = \Phi^{-1}(\alpha)\,, \tag{11}$$

Because the normal distribution is symmetric around its mean, then the previous expression can be rewritten as

$$\phi^{-1}(\alpha) = -\phi^{-1}(1-\alpha)\,, \tag{12}$$

Plugging the previous expression into (7) yields the formula of the value-at-risk with drift adjustment

$$VaR_{ht,\alpha} = \phi^{-1}(1-\alpha)\sigma_{ht} - \mu_{ht}\,, \tag{13}$$

where μ_{ht} represents the drift adjustment.

Under the assumption that the portfolio's expected return is the risk-free rate, $\mu_{ht} = 0$, and dropping the implicit dependence of VaR on time t, the previous expression can be further simplified as

$$VaR_{h,\alpha} = \phi^{-1}(1-\alpha)\sigma_{h}\,, \tag{14}$$

Under the assumption that the returns follow a normal distribution, the h-day VaR can be obtained resorting to the square root scaling rule, that is

$$VaR_{h,\alpha} = \sqrt{h} * VaR_{1,\alpha}\,, \tag{15}$$

2.3 Markov Decision Process

Markov decision processes, or simply, MDP's, are comprised of a set of spaces $\mathcal{S}$, a vector of possible actions $\mathcal{A}$, a transition and reward models. Miranda & Fackler (2002) sum up a MDP by a choice to be taken at each time step t, a_t, from the available relevant action set for the present state s_t, $\mathcal{A}(s_t)$, earning a reward R_t , from a function parametrized by both current state and selected action.

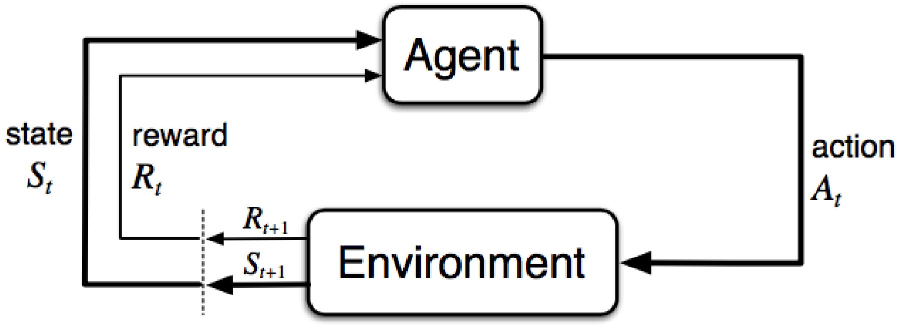

Figure 1 – The figure shows the agent-environment interaction in a Markov decision process, where the agent interacts with the environment by performing the selected action, and the environment returns the reward and new state associated with the agent's behavior.

The transition model defines which state the environment transitions to, contingent on the present state and selected action,

$$\mathcal{P}_{ss'}^{a} = P(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]\,, \tag{16}$$

where $\mathbb{P}$ denotes a probability function.

The reward function on the other hand, defines the reward yielded at time step t, contingent on the current state s and selected action a

$$R_t = R(s_t, a_t)\,, \tag{17}$$

These two components, the transition and rewards models, form the basis of a Markov decision process.

Markov introduced the concept of memorylessness of a stochastic process, a key propriety of MDPs. This assumption states that each state pair is independent of past-occurrences, meaning, each state contains all the meaningful information from the history. In other words, the future and past are conditionally independent given the present, since the current state contains all the required statistical information to decide upon the future.

The concept of policy, represented by the symbol π, which corresponds to a mapping from states to actions, is meaningful throughout the course of this work. Policies can be deterministic or stochastic. The former corresponds to a policy in which the optimal action is solely determined by the current state. The probability of an action being selected in state s under a deterministic policy π is given by

$$\pi(s) = a\,, \tag{18}$$

whereas if the policy π is stochastic, the action selection is contingent on a probability function

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]\,, \tag{19}$$

Value functions, $v_\pi(s)$ attribute a quantifiable goodness value to a given state under a policy π, by attempting to capture its associated future reward. The latter corresponds to the sum of discounted future rewards, or expected return, given by G_t

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k\,, \tag{20}$$

where $\gamma \in [0,1]$ represents the discount term, a factor by which to penalize future rewards.

The state-value function of a given state s at time t, corresponds to the expected reward the algorithm is to observe starting from state s

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]\,, \end{aligned} \tag{21}$$

where $\mathbb{E}_\pi[\cdot]$ represents the expected value of a given random variable considering the agent follows policy π.

Whilst the value function focuses on individual states, the action-value function q_π, seeks attributing a fitness value for the action as well,

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]\,, \end{aligned} \tag{22}$$

Using the probability distribution over all possible actions and q-values, that is, state-action values, the value function can be obtained by

$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} q_\pi(s,a)\pi(a|s)\,, \tag{23}$$

The overall objective under the MDP framework is to compute the value function which maximizes the rewards, $v_{\pi_*}(s) \geq v_\pi(s)\ \forall\ \pi, s$,

$$v_*(s) = \max_\pi v_\pi(s) \tag{24}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a)\,, \tag{25}$$

The previous goals can be achieved via Bellman's optimality equations

$$\mathrm{v}_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{26}$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a]\,, \tag{27}$$

The reinforcement learning models and concepts presented and used throughout this thesis, assume problems described by infinite horizon, stochastic transition model MDPs.

2.4 Proximal Policy Optimization

The current section presents, in a concise manner, the main ideas behind the proximal policy optimization algorithm, on which the agent applied in this thesis is based. For those unfamiliar with the topic, Appendix A attempts to shed light on the deep reinforcement learning algorithm taxonomy, starting with familiar concepts such as dynamic programming and Monte Carlo returns, to the present section's algorithm's predecessor, trust region policy optimization (TRPO). Note however, that the appendix merely scratches the surface in terms of reinforcement learning's literature. It is important to mention that there are several important and widely used algorithms not present in the referred section, which form the cornerstone of deep reinforcement learning. For instance, asynchronous advantage actor-critic (A3C) and its synchronous and deterministic version (A2C), deterministic policy gradient (DPG), deep deterministic policy gradient (DDPG), both of which, model the policy as a deterministic decision, distributed distributional DDPG (D4PG), actor-critic with experience replay (ACER), actor-critic using Kronecker-Factored trust region (ACKTR), soft actor-critic (SAC). However, in order to keep the algorithmic literature review as concise as possible, these have been left out of the present discussion. These algorithms form a clear timeline of deep reinforcement learning's evolution, thus, understanding them, may help clarifying and solidifying the employed concepts.

The trust region policy optimization algorithm introduced in Appendix A.III – Policy Gradient has a few shortcomings, namely, the computation of the Fisher matrix at every model parameter update, which is computationally expensive, and its requirement of large batches of rollouts to approximate the the Fisher matrix accurately.

Proximal policy optimization (Schulman, et al., 2017) is an on-policy algorithm, able to learn control problems under discrete or continuous action spaces, which seeks providing the answer for the same question as its predecessor, TRPO, does: to what extent can the policy be updated, using the available information, without modifying it too largely, which would result in performance collapse. Whereas TRPO provides the answer relying on the use of a complex second-order method, PPO is comprised of a set of first-order equations, combined with a few tricks, which ensure the similarity between the old and new policies.

Consider the probability ratio between the old and new policies

$$r(\theta) = \frac{\pi(a|s,\theta)}{\pi(a|s,\theta_{old})} \ , \tag{28}$$

Under the previous construct, TRPO's objective function can be represented as

$$J^{TRPO}(\theta) = E\left[r(\theta)\hat{A}_{\pi_{\theta_{old}}}(s,a)\right], \tag{29}$$

The previous equation represents another of TRPO's shortcomings not mentioned in the previous paragraph. Should the distance between policies not be bounded, then the algorithm would suffer from instability, caused by large parameter updates and large policy ratios. PPO imposes a constraint on this ratio, by ensuring $r(\theta)$ stays within the range $[1-\epsilon, 1+\epsilon]$, where ϵ is the clipping hyperparameter.

The new objective function, a surrogate function, is referred to as the Clipped[3] objective function, and is represented by $J^{CLIP}(\theta)$

$$J^{CLIP}(\theta) = E\left[\left(r(\theta)\hat{A}_{\pi_{\theta_{old}}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\pi_{\theta_{old}}}(s,a)\right)\right], \tag{30}$$

where $clip(\cdot)$ represents the clipping function, which ensures the first argument, is bounded by the remaining arguments.

The introduced objective function takes the minimum between the policy's ratio, and the clipped arguments, therefore preventing excessively large (θ) updates, especially when associated with very large rewards – which would result in extreme policy updates.

The version of proximal policy optimization employed throughout this thesis, corresponds to the algorithm's Clip or clipped version, under the actor-critic framework. In order to accommodate the actor-critic framework, the authors augmented the objective function with an error term on the value function, whilst adding an entropy term to encourage exploration.

$$J^{CLIP\prime}(\theta) = E\left[J^{CLIP}(\theta) - c_1\left(V_\theta(s) - V_{target}\right)^2 + c_2 H\left(s, \pi_\theta(a|s)\right)\right], \tag{31}$$

where c_1 corresponds to the error term coefficient, and c_2 to the entropy coefficient, both of which, are hyperparameters.

```
**Algorithm** Proximal Policy Optimization with Clipped Objective
1. Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2. **for** $k = 0,1,2,\ldots$ **do**
3.     Collect a set of partial trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi_{\theta_k}$ in the environment
4.     Compute rewards-to-go $\hat{R}_t$
5.     Compute advantages estimates, $\hat{A}_{\pi_{\theta_{old}}t}$, using any advantage estimation algorithm based on the current value function $V_{\phi_k}$
6.     Update the policy by maximizing the Clipped PPO objective:
           $\theta_{k+1} = \underset{\theta}{\operatorname{argmax}} \mathcal{L}_{\theta_k}(\theta)$
       by taking K steps of minibatch stochastic gradient ascent via Adam where
           $\mathcal{L}_{\theta_k} = \mathbb{E}_{t\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min\left(r_t(\theta)\hat{A}_{\pi_{\theta_{old}}t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\right)\right]\hat{A}_{\pi_{\theta_{old}}t}\right]$
   **end for**
```

Algorithm 1 – Proximal policy optimization pseudo-code.

[3] The function is not named PPO's objective function, as the algorithm introduces two versions of the objective function, PPO-Penalty and PPO-Clip. However, the author opted for only introducing the latter as it corresponds to the version used in this thesis.

3 Literature Review

Since the early 1990's, Value-at-Risk has increasingly become the standard for risk measuring and management throughout a wide range of industries. However, the concept attained special relevance with the 1995 amendment to the first Basel Accord, which enabled FI's to develop and use their own internal models, as described in section 2.1. Ever since, the literature on value-at-risk, ranging from its estimation to its optimization under the Basel Accord framework, has grown uninterruptedly. The present chapter focuses on introducing past findings on the subject at hands and summarizing the corresponding author's findings.

In 2007, Pérignon et al. (2007) introduced the notion that commercial banks tend to over-report their VaR estimate. The authors found three possible justifications for this fact. Firstly, incorrect risk aggregation methods induce higher VaR estimates by improperly accounting for the diversification effects. Secondly, banks tend to overstate their VaR metric to protect their reputation, as any evidence that the institutions are incapable of accurately measure their risk, would result in market-driven penalties. Lastly, a typical principal-agent problem takes place, where the risk manager voluntarily increases the VaR estimate to avoid attracting unwanted attention. Such behavior not only represents higher costs for the banks in terms of larger allocated capital and corresponding opportunity costs, but to the economy as well. Specifically, according to the author, the tendency to over-report risk leads to an exaggerate estimate of a bank's implicit risk from an investor's point of view, influencing its asset-pricing through the increase of required return on equity thus generating market distortions; furthermore, the excessive capital allocation leads to the rejection of funding for relevant projects, creating a loss of value for the economy.

With the introduction of the second Basel Accord, McAleer (2008) suggested a set of ten practices which aimed at helping FIs monitor and measure market risk, in order to minimize the daily capital charge, in particular when the FI focuses in holding and moving cash, rather than on risky financial investments. The author provided insights on (a) choosing between volatility models; (b) the underlying distribution's kurtosis and leverage effects; (c) the covariance and correlation relationship models; (d) the use of univariate or multivariate models to forecast the value-at-risk; (e) the underlying distribution's type selection according to the volatility type (conditional, stochastic or realized; (f) optimal parameter selection; (g) assumption derivation; (h) forecasting model's accuracy determination; the penultimate point,

(j) optimizing the FI's exceedances, which constitutes the object of this thesis; and lastly, (k) exceedance management and its relationship with the public interest.

In the aftermath of McAleer (2008)'s findings and recommendations on risk measurement and daily capital charge optimization, McAleer et al. (2009) introduced a model which attempted to design a rule to minimize the daily capital charges for institutions working under the Basel II regulation. This methodology, entitled the dynamic learning strategy (DYLES), was characterized by discrete and fast reactions whenever exceedances were recorded, being context sensitive in the sense that it accommodated past information, or violation history, onto its estimate, behaving more cautiously or conservatively when more exceedances had been recorded, and aggressively otherwise.

The authors formalized DYLES as:

$$\min_{\Theta=[P_0,\theta^P,\theta^R]} \frac{1}{250}\sum\nolimits_{p=j}^{250} \max\left[-P_t VaR(t-1), [3+k]\frac{1}{60}\sum\nolimits_{p=1}^{60} -P_t VaR(t-p)\right], \qquad (32)$$

where P_t is a variable which is tied to the number of recorded exceedances, representing how aggressive or conservative the estimate is regarding the estimated risk measure, the value-at-risk; θ^P and θ^R are, respectively, the penalty per violation and the 25-day accumulated reward; *p* represents the backtesting time step, from 1 to 250; and *k* represents the applicable multiplier increment, applied to 3, from the value set $[0.0; 0.4; 0.5; 0.65; 0.75; 0.85; 1]$, should there be more than 4 recorded exceedances in the previous financial year. Lastly, *t* represents, as usual, the date in which the system is being modeled in. The authors concluded that in all occasions, using the DYLES strategy led to higher capital savings when compared to a passive behavior, that is, disclosing the computed VaR, decreasing the capital requirements by 9.5%, on average, up to 14%, and reducing, on average, the number of violations from 12 to 8.

Kuo et al. (2013) proposed a modification to the DYLES decision rule, attempting to incorporate the challenges faced by banks created by the Basel III reforms as noted by Allen et al. (2012). In order to do so, Kuo et al. (2013) added the current market cost of capital to the DYLES rule, naming it, MOD-DYLES. The improved model not only allowed higher cost savings when compared with the original DYLES methodology, but it also enabled its extension to other VaR computation methods (Kuo, et al., 2013). The authors analyzed three strategies – standard unmanipulated disclosure, DYLES-based approach and the proposed MOD-DYLES model – for two VaR estimation methods, the constant variance-covariance, and the variance-covariance GARCH methods. For the first estimation method, Kuo et al. (2013) observed that

MOD-DYLES introduced an average saving of 1.36% when compared to the passive standard disclosure approach, whilst drastically decreasing violations. Focusing on its predecessor, the modified algorithm managed to save, on average, an extra 0.10% per year, although as expected given their identical underlying exceedance constraints, the number of recorded violations matched. Regarding the second estimation model, the VC-GARCH, the proposed framework saved on average 1.09% and an extra 0.06% per year, respectfully, for the standard disclosure, in comparison with DYLES.

Seixas (2016) sought to further exploit the concept first introduced by McAleer (2008) that the path to optimization should not focus exclusively on the VaR estimation method, but also on the percentage of the metric which should be disclosed, as a form of capital charge minimization. The researcher's work emphasized the need to construct an optimal policy for the daily VaR disclosure, under the internal model approach, as proposed by the Basel II Accords. In order to do so, Seixas (2016) applied the dynamic programming methodology to the problem at hands, using a discrete MDP with infinite periods. The author managed to create a policy which minimized the daily capital charge, considering (a) the time remaining until the back process takes place, (b) the number of recorded exceedances until the moment of decision, and lastly, (c) the applicable multiplier as defined by the preceding financial year's backtesting process. The policy generated tended to underreport the one-day value-at-risk, that is, to adopt an aggressive approach, which is consistent with McAleer et al. (2009)'s notes on risk manager's behavior. The use of the dynamic programming framework allowed surpassing some of the limitations and obstacles faced by DYLES's, namely, its employment being contingent on the portfolio at hands, and the parameter estimation problem, which would represent a different disclosure rule contingent on the underlying distribution. When compared to the standard option of disclosure based on a normal distribution's strategy, Seixas (2016)'s proposal outperformed the normal strategy in 82% of the times, yielding an average saving of 7.22% per day, when applied to a S&P500 portfolio.

4 Methodology

The purpose of the current chapter is twofold. Firstly, it models the Basel framework introduced in section 2.1 into a Markov decision process suitable for deep reinforcement learning. Secondly, the author covers the introduction and assessment of an agent, based on proximal policy optimization under the actor-critic framework, which attempts to approximate Seixas (2016)'s solution to the Basel problem.

4.1 Environment Model

The problem at hands has been thoroughly introduced in chapter 2.1. The current section turns to formalizing the environment so it can be solved via DRL.

Recall that the model, under the model-free reinforcement learning framework, is comprised of three components, the state and action spaces, and the reward function. The agent's goal under the proposed environment, is to minimize the daily regulatory capital charge ($k * x * VaR_t$) by optimizing the percentage of the value-at-risk to disclose, x. The problem represents a trade-off between opting to minimize the short-term disclosure – thus obtaining larger daily rewards – and the long-term cost, in the shape of the applicable multiplier, a direct function of the incurred exceedances, which in turn, are partly[4] dictated by the value-at-risk manipulation.

The state-space is characterized by three key aspects: (a) the time remaining until the backtesting process takes place (TtoB); (b) the number of recorded exceedances in the current trading period (EC); and lastly, (c) the applicable multiplier (K) throughout the given period. The first constituent, TtoB, corresponds to how much time, measured in days, remains until the new regulatory backtesting process starts, that is, before the year's disclosure is accounted for, and its quality assessed, considering 250 trading days. The variable's natural behavior is then to decade from its maximum value, until 1, when the review procedure takes place. The second flag, represented by the acronym EC, accounts for the number of exceedances recorded in the current episode. The set of possible values for this variable starts at 0, where no exceedance has been registered, and its maximum value is set at 11. The reasoning for the limitation to 11 exceedances is due to the fact that any further violations would still place the

[4] The idea of partial responsibility in the action selection in determining the next period's applicable multiplier is due to the stochastic nature of the environment's transitions.

FI on the highest multiplier, hence, rational behavior dictates that at this point, the institution would report a null value-at-risk, so as to decrease the short-term costs. However, the environment is modeled under the assumption that reaching the 10-exceedance threshold, would translate in the FI not being able to deploy its internal model, being forced by the regulator to use a standard risk model instead. For this reason, the agent is incentivized to avoid the 10th exceedance at all costs, being forced to report the maximum disclosure value, under the assumption that reaching said threshold would imply the FI incurring in high reputational costs. The third variable's values correspond to those defined in table 1. Accordingly, the state space corresponds to a 3-dimensional vector, comprised of the three discrete variables defined above, that is, one where each observation is a tuple comprised of one element of each discrete group. Specifically, $\mathcal{T} = \{\tau \mid \tau \in \mathbb{Z}, 1 \leq x \leq 250\}$, $\mathcal{E} = \{\epsilon \mid \epsilon \in \mathbb{Z}, 0 \leq x \leq 11\}$ and $\mathcal{K} = \{3, 3.4\,, 3.5\,, 3.65\,, 3.75, 3.85, 4, 10000\}$, which correspond to the time, exceedance and multiplier spaces, respectively. Accordingly, an observation under the current environment is defined as

$$s_t = (\tau \in \mathcal{T}, \epsilon \in \ \mathcal{E}, \kappa \in \mathcal{K})\,, \tag{33}$$

Note, however, the inclusion of an additional multiplier in $\mathcal{K}$, corresponding to 10000, which corresponds to a bankruptcy state, thus being represented by an extreme value, one which the institution would avoid at all cost.

In turn, the action space comprises the possible disclosure values, in percentage of the reported value, corresponding to values in the interval $]0, 3]$ with increments of $1E^{-3}$, that is,

$$\mathcal{A} = \{n/1000 \mid n \in \mathbb{N}^{+}, n \leq 3000\}\,, \tag{34}$$

where reporting the space $\mathcal{A}$'s ceiling corresponds to a situation where VaR disclosure is threefold, and vice-versa.

Despite being a model-free algorithm, a simulator must be present to be able to generate new samples from which the agent can learn. Such simulator describes the transitioned state s_{t+1}, contingent on the current state s_t and the selected action a_t, $\mathcal{P}_{ss'}^{a}$. For the purpose of solving the described problem, a simple transition simulator based on Seixas (2016) has been used. Such entity is governed by the table which follows

Event	**Probability**
No Exceedance	$P(Z < VaR * x)$
Exceedance	$1 - P(Z < VaR * x) - P(Z > Capital\ Charge)$
Bankruptcy	$P(Z > Capital\ Charge)$

Table 3 – Transition probabilities for the simulator

where Z is a standardized normal distribution of profit and loss, under which, the probabilities are yielded by

$$P(Z < VaR * x) = \Phi\left(x * VaR_{\alpha,h}, \mu, \sigma\right) \tag{35}$$

$$P(Z > Capital\ Charge) = 1 - \Phi\left(k_t * VaR_{\alpha,h} * \sqrt{10}, \mu, \sigma\right), \tag{36}$$

One of the critical aspects of reinforcement learning is the reward function. Such construction is target of a both feature engineering, and prior domain knowledge. A surrogate reward function is introduced to help the algorithm converge to the optimal policy faster and more accurately, given its episodic nature. Specifically, an additional reward is given when the backtesting process occurs. Mathematically, $f(a_t, s_t)$ is given by

$$f(a_t, s_t) = f(a_t, [\,ttob, ec, k]_t) = -k_t * a_t * VaR_{1,0.01} * \sqrt{10} + I_t(s_t) * \mathcal{R}(s_t), \tag{37}$$

where $VaR_{1,0.01}$ represents the one day value-at-risk computed at the 1% significance level, and $I_t(s_t)$ is an indicator function, which takes the value one if the episode's time remaining to backtesting (Ttob) equals one, and zero otherwise,

$$I_t(s_t) = \begin{cases} 1\, if\ TtoB_t = 1 \\ 0\, if\ TtoB_t = 0 \end{cases}, \tag{38}$$

and $\mathcal{R}(s_t)$ corresponds to the terminal reward space,

$$\mathcal{R}(s_t) = -10E3 * \begin{cases} R(k=0) + (K_i - K_{i+1})/2\,, & 1 \leq k \leq 6 \\ 0.1, & k = 0 \\ 0, & k = 7 \end{cases}, \tag{39}$$

Applying the previous equation, yields the eight additional reward factors

$$\mathcal{R}(s_t) = [0.1, 0.3, 0.35, 0.425, 0.475, 0.525, 0.6, 0] * (-10E3), \tag{40}$$

The introduced surrogate reward aims at accelerating learning by injecting domain knowledge directly into the reward space, filling in the gap left by the use of episodic reinforcement learning (RL). By doing so, the long-term objective is hardcoded into the agent which would otherwise be unperceived, as under episodic RL the agent never reaches the next period where the multiplier revision is materialized. The additional reward space attempts to capture the multiplier distribution's underlying structure, by measuring the difference between multipliers, and scaling the gradient to the reward distribution's parameters, a behavior similar to that of McAleer et al. (2009). Notice that the last multiplier, associated with bankruptcy, has a null reward, which is counterintuitive, as rationale dictates such state should be associated with additional penalties – the additional rewards are negative, according to the standard reward distribution. The reasoning behind such formulation, is that the reward for defaults is extremely negative, hence, penalizing the behavior further will only hinder policy gradients by contributing to the occurrence of exploding gradients, whilst producing no additional information. However, the additional space introduces a bias towards exceedances up to 4 (with the lowest multiplier), as the agent may seek exploiting this lower penalty - or higher reward. Yet, due to the small number of iterations, the author saw fit adding it to accelerate learning. For researchers aiming to solve the problem with more iterations, it is not advisable to use the proposed modification without extensive testing.

Note that during training, the reward function has been scaled to the [-1, 0] range, to prevent exploding gradients and to ease gradient descent's functioning, as is typical in DRL literature.

The algorithm's network is the same for both actor and critic, 2 hidden layers, each containing 64 neurons, the first layer using a Rectified Linear Unit (ReLu) (Agarap, 2019) activation function for both entities. Activation functions are needed to create non-linear transformations, as a neural network is only capable of performing linear transformations without non-linear activation functions. To understand more deep learning concepts such as neural networks and their architectures, the reader is advised to delve into Goodfellow et al. (2016).

Deep learning algorithms can prove troublesome to understand and implement, particularly for those delving in the class of algorithms for the first time. For this reason, and to remove some of the conceptual abstraction, the remainder of the present section attempts to represent in simple terms the operations performed in a simple version of PPO, that is, to create an implementation pseudo-code. In practice, typical implementations rely on the use of Generalized Advantage Estimators (GAE), multithreading, among other useful modifications to simplify and speed computations and help convergence. These have been omitted from the representation. Yet, to demonstrate the easiness of the algorithms in accommodating a variety of situations, the scheme presents the process for both discrete and continuous actions.

Algorithm Actor Critic PPO Implementation

```
1.      Define the learning constants
1.1         $E\_MAX$ : Maximum number of episodes
1.2         $E\_LEN$ : The episode's length
1.3         $\gamma$ : The discount factor $\gamma$
1.4         $\alpha_A$ : Actor's Learning Rate
1.5         $\alpha_C$: Critic's Learning Rate
1.6         $MIN\_BATCH\_SIZE$ : Minimum batch size for updates
1.7         $A\_UPDATE\_STEP$S: Actor's update operation n-step loop length
1.8         $C\_UPDATE\_STEP$S: Critic's update operation n-step loop length
1.9         $\epsilon$: Surrogate objective function clip term $\epsilon$
1.10        $S\_DIM$ : State space shape
1.11        $A\_DIM$ : Action space shape

2.      Initialize the critic:
2.1         Input layer: Dense layer
                input: $S\_DIM$
                shape: [64, 1]
                activation: ReLu
                trainable: True
2.2         Hidden Layer: Dense Layer
                input: Input layer (2.1)
                shape: [64, 1]
                activation: ReLu
                trainable: True
2.3         Value Estimation Layer - Output Layer: Dense layer
                input: Hidden Layer (2.2)
                shape: 1
                activation: None
                trainable: True

3.      Define the critic's Optimization Function:
3.1         $\hat{A}_\theta(s,a) = r - V(s)$
3.2         Define the loss function: $\hat{A}_\theta(s,a)^2$
3.3         Optimization function c_trainop:
                Adam Optimizer: learning rate: $\alpha_C$
                minimization target: 3.2

4.      Define the actor ($\pi$):
4.1         Input layer: Dense layer
                input: $S\_DIM$
                shape = [64, 1]
                activation = Relu
                trainable = True
4.2         Hidden Layer: Dense layer
                input: Input layer (4.1)
                shape: [64, 1]
                activation: ReLu
                trainable: True
```

```
4.3          Action ($\pi$) Layer - Output Layer : Dense Layer
                    input: Hidden Layer (4.2)
                    shape: A_DIM
                    activation: Softmax
                    trainable: True

5.           Define the old actor ($\pi_{old}$):
5.1             $\pi_{old} = \pi$, only the former is not trainable

6.           Define the Actor's Optimization Function
6.1             Compute the probability ratio $r(\theta) = \frac{\pi(a|s,\theta)}{\pi(a|s,\theta_{old})}$
6.2             Compute the Surrogate Loss $r(\theta) * \hat{A}_{\theta}(s, a)$
6.3             Compute PPO's Clipped Loss = $-\min\left(r_t(\theta)\hat{A}_t^{\pi_k}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\right)$
6.4             Optimization Function a_trainop:
                    Adam Optimizer: learning rate: $\alpha_A$,
                    minimization target: (6.3)

7.           Define the Update Function
7.1             Receive input: state $s$, action $a$, reward R
7.2             Replace $\pi_{old}$ with $\pi$
7.3             Compute $V(s)$ using the Critic (2.)
7.4             Compute the advantage function (3.1)
7.5             Update the Actor using PPO's clipping method: execute a_trainop for A_UPDATE_STEPS
7.6             Update the Critic: execute c_trainop for C_UPDATE_STEPS

8.           Run the Simulation
8.1             Build the environment env
8.2             for ep in E_MAX
8.2.1               Reset the environment and observe S
8.2.2               Define the state, action and reward buffers $\mathcal{D}_s, \mathcal{D}_a, \mathcal{D}_r$
8.2.3               for $t$ in E_LEN:
8.2.3.1                                         $a = \pi(s)$
8.2.3.2                 Perform $a$ and observe S' and R
8.2.3.3                 Add state, action and reward to the corresponding buffers
                                                $\mathcal{D}_s += s$
                                                $\mathcal{D}_a += a$
                                                $\mathcal{D}_r += R$
8.2.3.4                 Set $S' = S$
8.2.5.5                 If $length(S) \geq MIN\_BATCH\_SIZE$:
8.2.5.5.1                   Compute $V(S)$ using the Critic (2.)
8.2.5.5.2                   Compute the $V(s_t)$ for each reward in the buffer, inserting it into a buffer
                                                $V_t = r_t + \gamma V_{t+1}$
                                                $D_{r^{disc}} += V_t$

8.2.5.5.3                   Update PPO (7.) using $\mathcal{D}_s, \mathcal{D}_a, \mathcal{D}_{r^{disc}}$

8.2.5.5.4                   Set $\mathcal{D}_s = \mathcal{D}_a = \mathcal{D}_r = \mathcal{D}_{r^{disc}} = Empty$
8.2.4               end for
8.3             end for
```

Algorithm 2 – Actor Critic PPO implementation pseudo-code.

Throughout this thesis numerous advantages of using DRL versus DP have been appointed, with section 4.2.6 providing a more exhaustive list of the advantages of using the current class of algorithms. One of which, is the ease to adapt and modify the model's architecture. The previous pseudo-code illustrates how both the actor and critic's structure can be changed by simply plugging additional hidden layers, which can be done programmatically. Another mentioned key aspect is the ability to include continuous state and actions spaces. In fact, the inclusion of a continuous action space estimation merely involves a small change to the critic's architecture, requiring the underlying distribution's parameters – in the following case, a univariate Gaussian's mean and standard deviation - to be optimized, instead of the critic's network π. Specifically, the point 4. In the previous pseudo-code would be modified to the following

Algorithm Actor Critic PPO Implementation – Actor under a Gaussian probability distribution

```
4.      Define the actor ($\pi$):
4.1       Input layer: Dense layer
              input: S_DIM
              shape [64, 1]
              activation: ReLu
              trainable: True
4.2       $\mu$ Layer: Dense layer
              input: Input layer (4.1)
              shape: $A\_DIM$
              activation: Tahn
              trainable: True
4.3       $\sigma$ Layer: Dense layer
              input: Input layer (4.1)
              shape: $A\_DIM$
              activation: Softplus
              trainable: True
4.4       return $N \sim (\mu, \sigma)$
```

Algorithm 3 – A modification to Algorithm 2 to encompass continuous actions.

4.2 Results

The purpose of the current work is not to yield a policy better than that of Seixas (2016)'s, given it represents an optimal policy, nor to fully reproduce the author's result, but is instead, to demonstrate the capabilities of deep reinforcement learning in approximating a solution to the same problem, whilst bypassing some dynamic programming's hindrances. Hence, the section will be divided into three parts. The first section will focus on the computational requirements used in training the model, whilst providing a few insights into the process of estimating an increasingly optimal policy under the DRL framework. The second section shifts the attention towards analyzing the agent's behavior and learning capabilities through the evolution of training metrics as a function of the agent's training iterations. The subsequent chapter then assesses the convergence and yielded policy similarity to that of Seixas (2016)'s through a shallow statistical analysis, given the latter represents an optimal policy for the problem at hands. Finally, an evaluation of the estimated policy's performance under a Monte Carlo simulation is made, in comparison with both Seixas (2016)'s and the non-manipulative strategies.

4.2.1 Computational Considerations

The results produced by an optimization algorithm which rely on the iterative method are greatly dictated by the amount of used iterations[5] - the terms iteration and training iteration are used interchangeably throughout this thesis. Reinforcement learning is no different. DRL applications for complex environments, are usually solved using learning iterations in the order of millions. Despite not classifying as a very complex problem, the Basel model being solved in the present work, is far from a Mountain Car or Inverted Pendulum[6] problem. As the chapter which follows demonstrates, only 70,000 iterations were used to train the model, far from the millions referenced in the previous sentence. The reasoning behind this decision falls back on financial constraints, as the training was fully funded by the author, from hyperparameter tuning – a critical aspect in deep models – to the actual training.

The hardware used to deploy and train the model consisted of 32 2.7 GHz Intel Xeon E5 2686 v4 CPUs, 2 NVIDIA Tesla M60 GPUs, with each GPU delivering up to 2.048 parallel processing cores and 8 GiB of GPU memory, hence, 16GB of GPU memory, and 244 GiB of

[5] The term learning iteration refers to a point in which the model's parameters are updated through the use of the environment samples, not the amount of complete sample trajectories – a full episode.

[6] The Inverted Pendulum and Mountain Car are two simple and classic environments used to benchmark reinforcement learning algorithms, typically solved in a few hundred learning iterations under agents with appropriate hyperparameter tuning and algorithm.

RAM[7] with an approximate cost of 2.16€ per hour[8] [9]. Considering each training iteration took, on average, 2.35 seconds, and 70,000 iterations were used, this represents an approximate cost of 355.32€, or 106.59€ when considering a 70% discount mentioned in footnote 9 and a training duration of approximately 45 hours. Under the specified conditions, training the model on 1,000,000 learning iterations would correspond to an investment of approximately 5,076,000€ or, 1,522,800€ under a 70% discount, and around 27 days.

The previous paragraph occluded hyperparameter tuning. In the case of proximal policy optimization, typical implementations contain around 16 hyperparameters – including actor and critic's network architecture, in depth and width – eligible for optimization. Such process greatly increases the cost associated with learning a policy via deep reinforcement learning, should optimal performance be an objective. This procedure can be accelerated should there be prior domain knowledge, as experienced researchers are able to tell which parameters are worth tuning, and in which range. Nonetheless, this additional prior step, is bound to consume as much or more resources than the actual training, in both time and expenses.

Before proceeding to analyzing the model's training progress, it is worthwhile mentioning that simpler models, such as DDQN – double dueling deep Q-network, a variant of the deep Q-learning model exposed in Appendix A.II – Deep Reinforcement Learning– might be more suitable for the problem at hands, since it corresponds to a discrete environment, where such algorithm has demonstrated sufficient convergence capabilities, requiring far less resources than PPO. However, the purpose of this thesis is to show that deep reinforcement learning, typically ignored by the financial research community, and policy gradient algorithms in particular, which unlock a variant of possibilities in the financial realm, display signs of strong convergence even for those with minimal knowledge on the subject – as is the author's case. For these reasons, the author opted for employing PPO.

[7] These resources are unnecessary when training the model. In fact, that model consumes little RAM, around 2GB, relying instead on the use of CPUs to asynchronously generate model samples, and GPUs to perform the computationally expensive linear algebra computations. Nonetheless, they constitute the employed hardware.

[8] In practice, when training the model one incurs in additional costs associated with storage, especially if the computation instance does not reside in the same location as the store instance. Nonetheless, such cost is negligible when considering the computational instance cost.

[9] Price reductions up to 70% are possible in several computation instance providers, by relying on the use of available instances whose price is dictated by supply and demand, much like financial markets.

4.2.2 Training Progress

As exposed in the previous chapter, the model's performance will be dictated not only by the specified hyperparameters, but largely by the number of performed learning iterations. Hence, assessing the algorithm's learning capability as a function of the iterations becomes a critical aspect of the present analysis. In order to do so, a few progress metrics have been selected. Note however, that observation metrics such as the average of the iteration's exceedances or multiplier do not qualify as robust metrics, as these are greatly influenced by both the episode's starting conditions, and the agent's randomness, materialized via entropy in policy gradient algorithms. The agent's cumulative reward – a standard metric in the reinforcement learning literature – and the average time remaining to backtesting have been selected as proxy performance metrics, whose behavior is predictable for converging agents. In a nutshell, the cumulative reward is expected to become increasingly positive – away from -1, the bankruptcy state – whilst the second metric is expected to converge towards 0.

As expected for a performing agent, the average scaled reward displays an upwards trend towards zero. This behavior corresponds to that of an agent which, on average, reports daily values which do not lead to bankruptcy, a sign of an appropriate reward function and long-term planning by the agent. The average reward exhibits an increasingly stable behavior with decreasing variance, which suggests a certain degree of convergence.

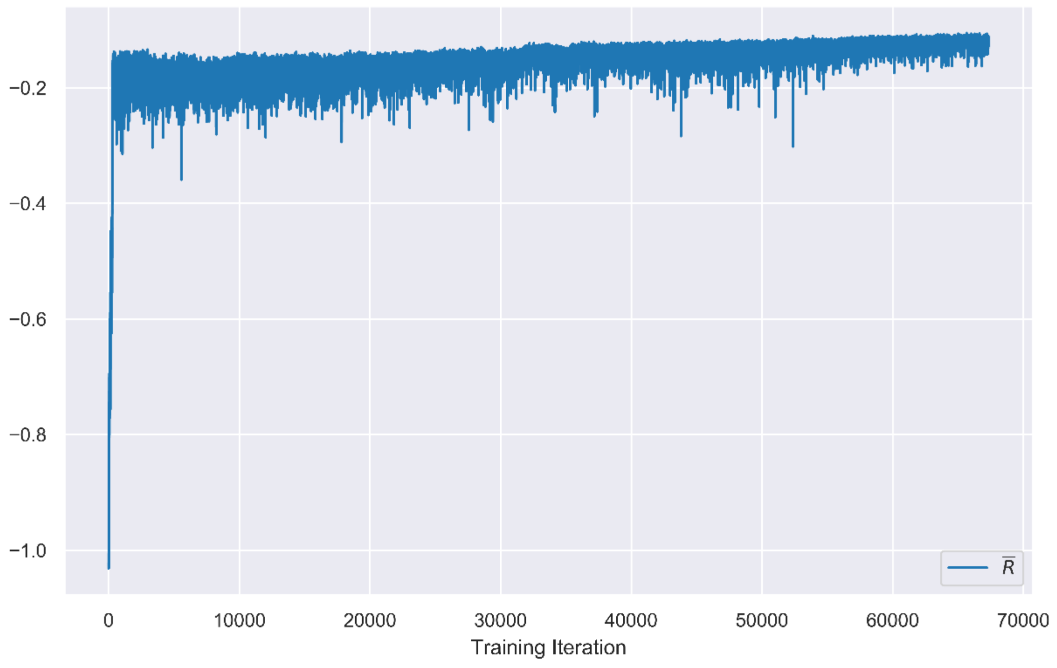

Figure 2 – The training process' average scaled rewards to the interval [-1, 0] as a function of training iterations.

The evolution of the average inverse of episode duration – time remaining to backtesting – decreases over time, exhibiting very small average durations on the first few hundred iterations, rapidly converging towards 0. These dynamics are consistent with that of an agent which understands the consequences of over-optimizing the short-term reward through very small disclosure, resulting in a bankruptcy state. The statement is compliant with figure 2, where the agent obtained large negative rewards beyond -1. Given the bankruptcy state's reward corresponds to -1, the fact that for such cases, the agent obtained, on average, rewards smaller than -1, is consistent with the episode duration analysis, as in the latter, the agent steers away from the minimal episode duration, thus accumulating increasingly negative rewards, which are added to the final state – bankruptcy – thus yielding a reward smaller than -1.

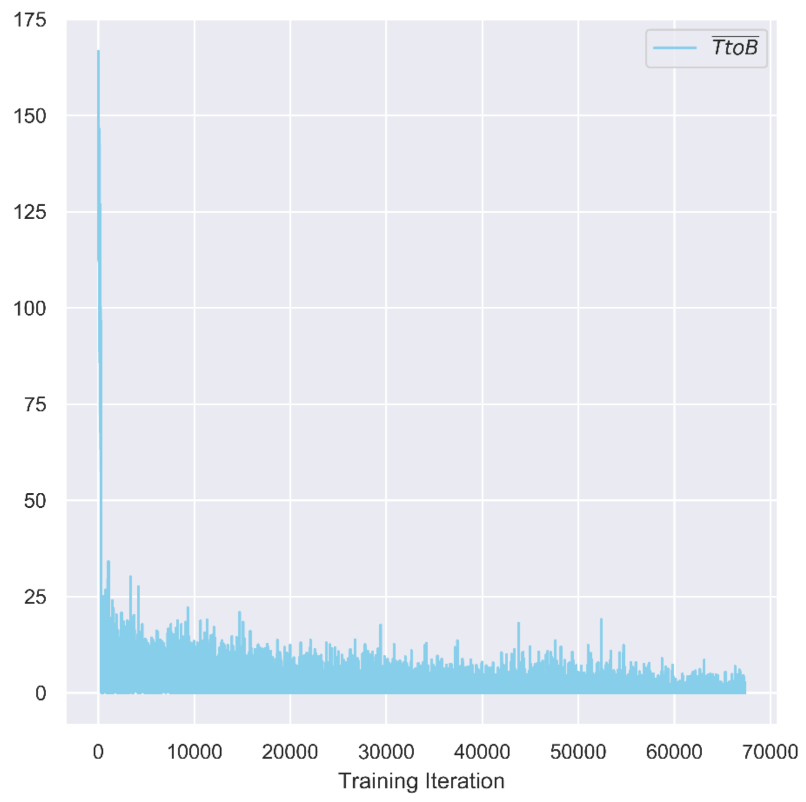

Figure 3 - The training process' average time remaining to backtesting, or inverse episode duration as a function of training iterations.

4.2.3 Policy Analysis

Given the produced policy is sub-optimal and has high variance, both of which, were expected, direct policy interpretation via graphs is not relevant, as a smooth policy was not learnt – see figure 6 and its discussion – and does not constitute this thesis' goal. Yet, and despite the simulated benchmark to be performed in the next chapter, the current chapter compares each policies' statistical indicators, to assess to what extent their outline differs.

Disclosure	RL	DP
Mean	0.955	1.016
Standard deviation	0.248	0.439
IQR	0.184	0.201
Q_1	0.832	0.860
Q_3	1.016	1.061

Table 4 - The main statistical indicators for the reinforcement learning and dynamic programming policy's disclosure, where IQR, Q_1 and Q_2 represent, the interquartile range, first and third quartiles, respectively. The disclosure is measured in the interval]0, 3] in accordance with equation (34).

An initial comparison between each policies' key statistical indicators suggests that the reinforcement learning policy reports, on average, lower percentages of the value-at-risk, regarding its counterpart, considering the entire policy. The artificial intelligence (AI) policy's statistical indicators exhibit smaller standard deviation and interquartile-range. The optimal policy's first quartile is larger than its AI counterpart, suggesting a less conservative behavior on the dynamic programming strategy, which is then compensated via higher disclosures, leading to a larger policy mean. Except for the last statement, these findings correspond to the opposite of the expected behavior. The previous table creates an illusion that the policy has converged, especially when combined with figure 4. The remainder of the current chapter will focus on explaining these results, and how they correlate with mean estimation convergence – typically referred to as convergence in mean – so as to unveil the actual convergence extent.

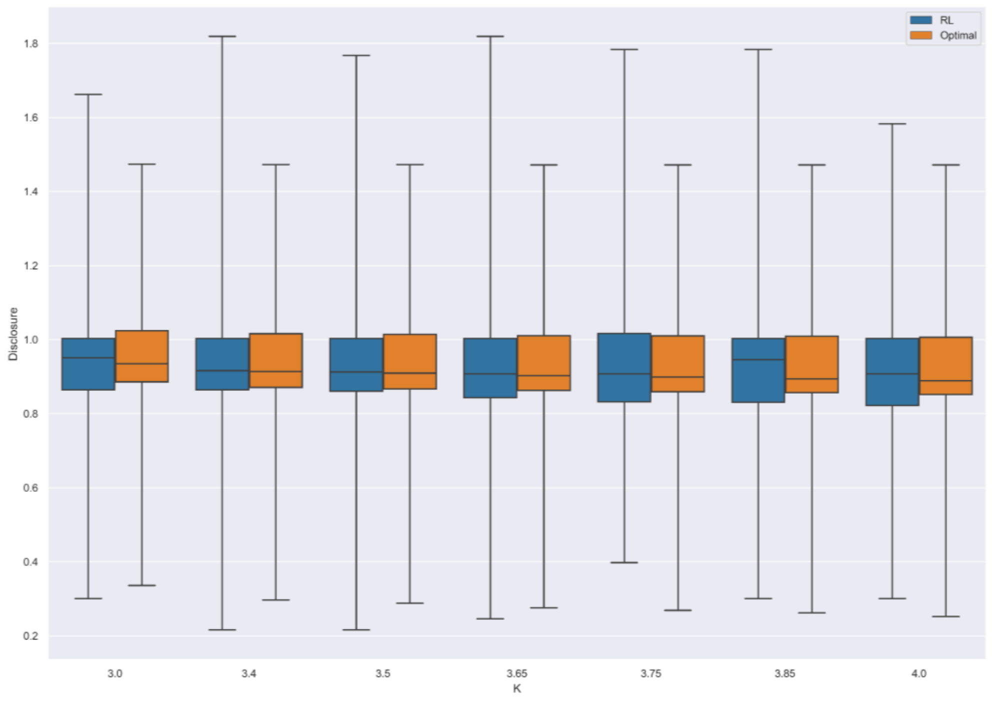

Figure 4 - Boxplot of the reinforcement learning and dynamic programming's policies' disclosure as a function of the multiplier index. The last multiplier, associated with bankruptcy, has been omitted.

Figure 4 depicts a boxplot of the estimated policies generated for each of the possible multiplier values – except for last multiplier, which corresponds to a state of bankruptcy in which the concept of disclosure no longer makes sense. The graph highlights the fact that the RL policy seems to converge towards the optimal policy, in mean. However, the former showcases a higher disclosure dispersion, characterized by higher presence of outliers, in both amount and value, on average, 20% higher than its DP counterpart, on the upper whiskers. The idea of mean estimation convergence can be further explored by analyzing the disclosure boxplot as a function of exceedances for a given multiplier.

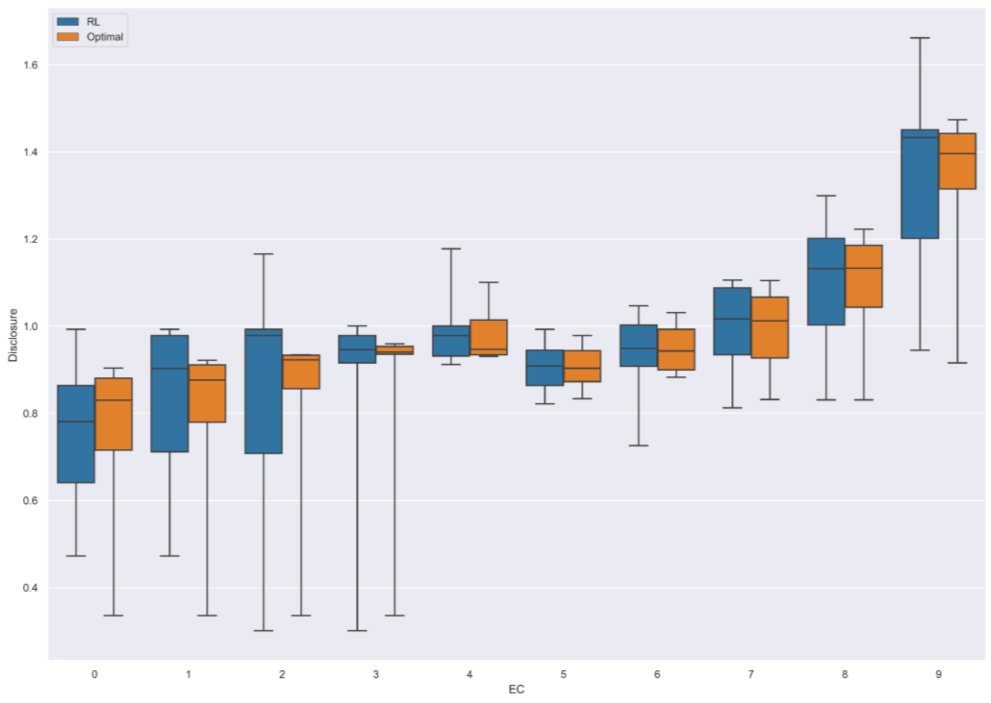

Figure 5 - Boxplot of the reinforcement learning and dynamic programming's policies' disclosure as a function of the number of exceedances, considering a multiplier equal to 3. The cases of 10 and 11 exceedances have been omitted

Figure 5 depicts a different behavior in terms of policy convergence. Not only do the policies exhibit differences in their disclosure mean per exceedance – considering the same multiplier – but the reinforcement learning policy exhibits longer whiskers, and for most cases, a larger interquartile range. Comparing with the overall policy's boxplot depicted in figure 4, whereas the RL's individual exceedance boxes display significant differences regarding the optimal policy's, the global boxplots present in the previous figure, in this case, considering the default multiplier, appear to converge. In other words, the policies appear to converge, in population mean, but not in sample mean.

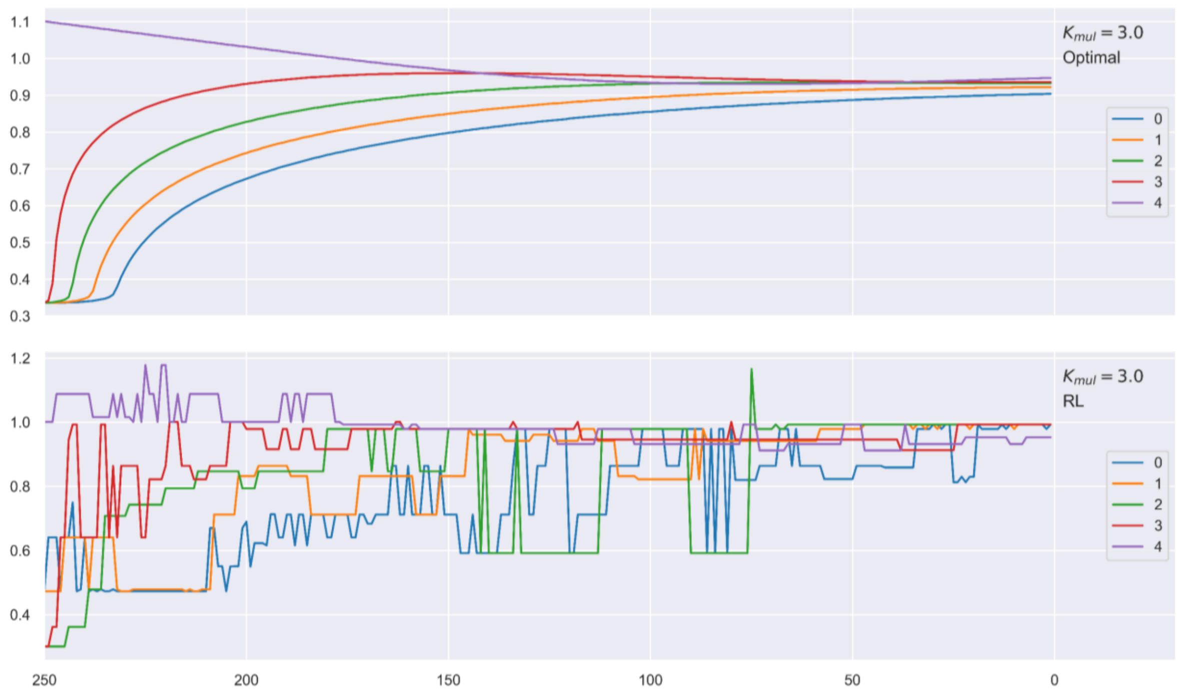

Figure 6 - The reinforcement learning and dynamic programming (optimal)'s trajectory as a function of the time remaining to backtesting, for exceedances between 0 and 4, considering the default multiplier.

Figure 6 represents each policies' trajectory as a function of the time remaining to backtesting, considering the multiplier equal to 3. Both policies exhibit the same overall behavior, underreporting in the start of the period, converging towards a baseline as the backtesting process approaches. Nonetheless, and despite the RL agent's apparent erratic behavior, they converge onto approximately the same value, and, overall, exhibit a similar increasing trend. Such behavior, analyzed and justified in section 4.2.1, is expected, given the small number of iterations.

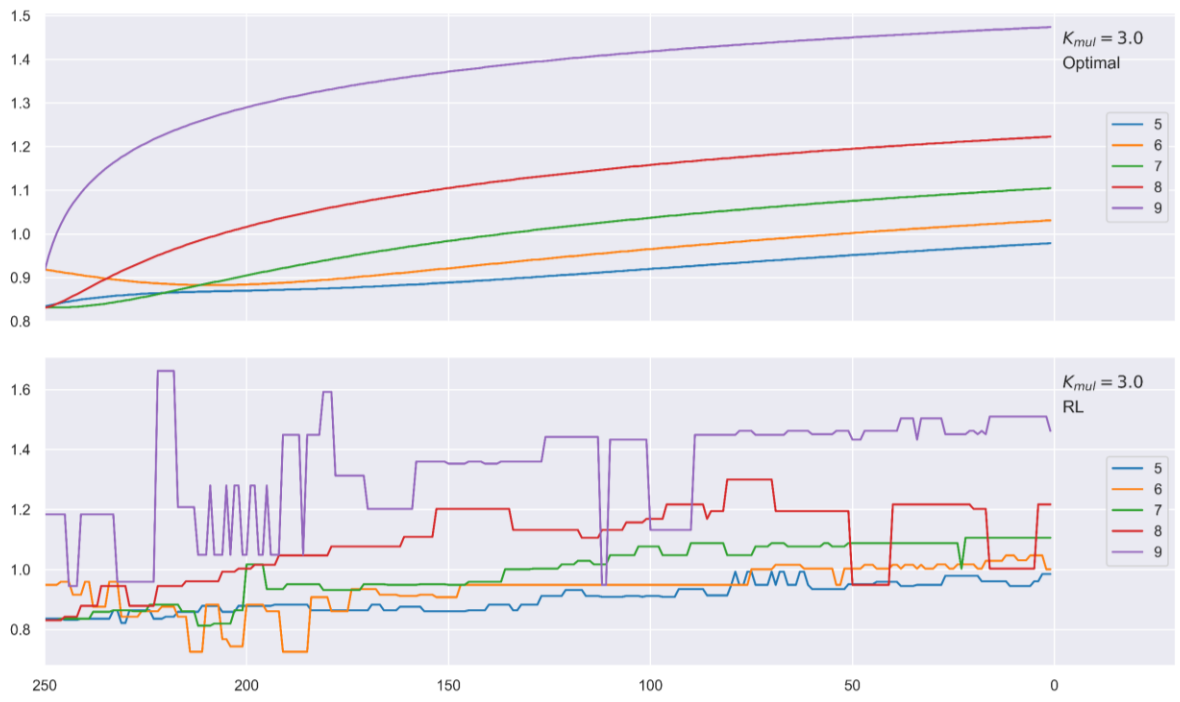

Figure 7 - The reinforcement learning and dynamic programming (optimal)'s trajectory as a function of the time remaining for backtesting, for exceedances between 5 and 9, considering the default multiplier.

The same pattern is observed in higher exceedance levels, where the RL policy exhibits a seemingly erratic behavior, though converging towards a point similar to that of its DP counterpart. Even so, the overall similarities between policies remain. Both agents disclose similar initial values, exhibiting akin smoothed trajectories through time. In fact, the pattern is observed throughout the entire policy, an indicator that the reinforcement agent began to grasp the environment's dynamics. The long-term trend exhibited by both policies, in conjunction with the RL policies' high variance behavior, sheds lights on the seemingly identical policy when analyzing the disclosure means in figure 4, and the relation with its long whiskers.

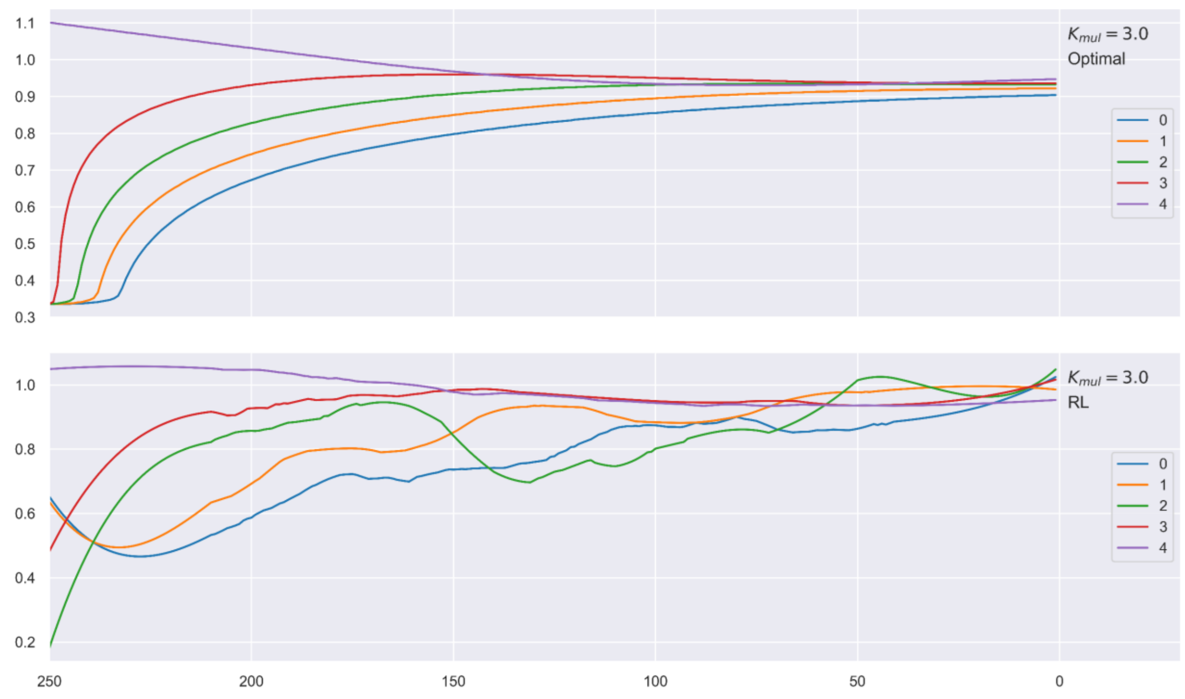

Figure 8 - The reinforcement learning and dynamic programming's trajectory, where the RL map was smoothed via a Savitzky-Golay filter, as a function of the time remaining to backtesting, for exceedances between 0 and 4, considering the default multiplier.

The presence of noise in the reinforcement learning policy hinders the perception of mean convergence. Figure 8 represents the same case as figure 6, only applying a Savitzky-Golay filter with 81 parameters and a polynomial of the 3rd order to help filter noise. The filtered image further supports the idea of mean convergence, as excluding noise, the RL policy appears to mimic Seixas (2016)'s policy, exhibiting a similar trend and overall pattern as time progresses, ultimately, converging towards values around 0.95 – which corresponds to reporting around 95% of the value-at-risk on the eve of the backtesting process, considering 3 as the applicable multiplier.

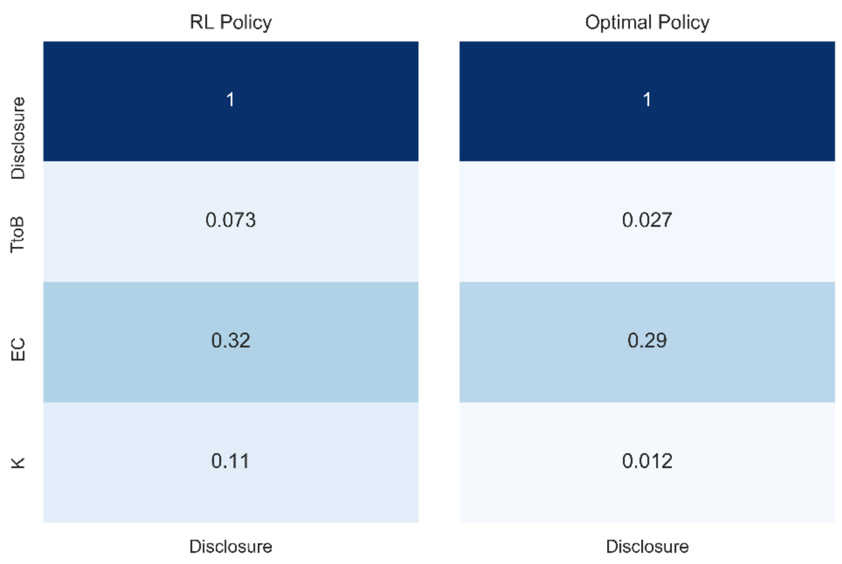

Figure 9 - The correlation between the disclosed value and the state's variables for the reinforcement learning and dynamic programming (optimal) policies.

Figure 9 represents the correlation coefficients for each policy. The relationship between the number of exceedances and the disclosed amount, seems to be conserved – 0.32 in the optimal policy, and 0.29 in the reinforcement learning policy. However, the positive correlation with the time remaining to backtesting and the applicable multiplier, represented by $TtoB$ and K, respectively, though small in Seixas (2016)'s, seems to have declined. This variance is particularly concerning in the case of the multiplier's correlation, reducing from 0.11, a low significance level in the optimal policy, to 0.012, a virtually nonexistent relation. This behavior is expected, as seen in the previous graphs, the RL policy depicts high variance in comparison with the optimal policy, the latter being characterized by smooth and continuous movements. This behavior leads to a smaller Pearson correlation index, hence, smaller correlation terms.

Recall the Pearson correlation index is given by

$$r_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}, \qquad (41)$$

where X and Y represent two jointly distributed real-valued random variables, σ_X and σ_Y correspond to X and Y's standard deviation, respectively, and $Cov(X,Y)$ represents the covariance between X and Y.

The larger the random variable's variance – and consequently, the standard deviation – the smaller the effect of their covariance on the Pearson correlation index.

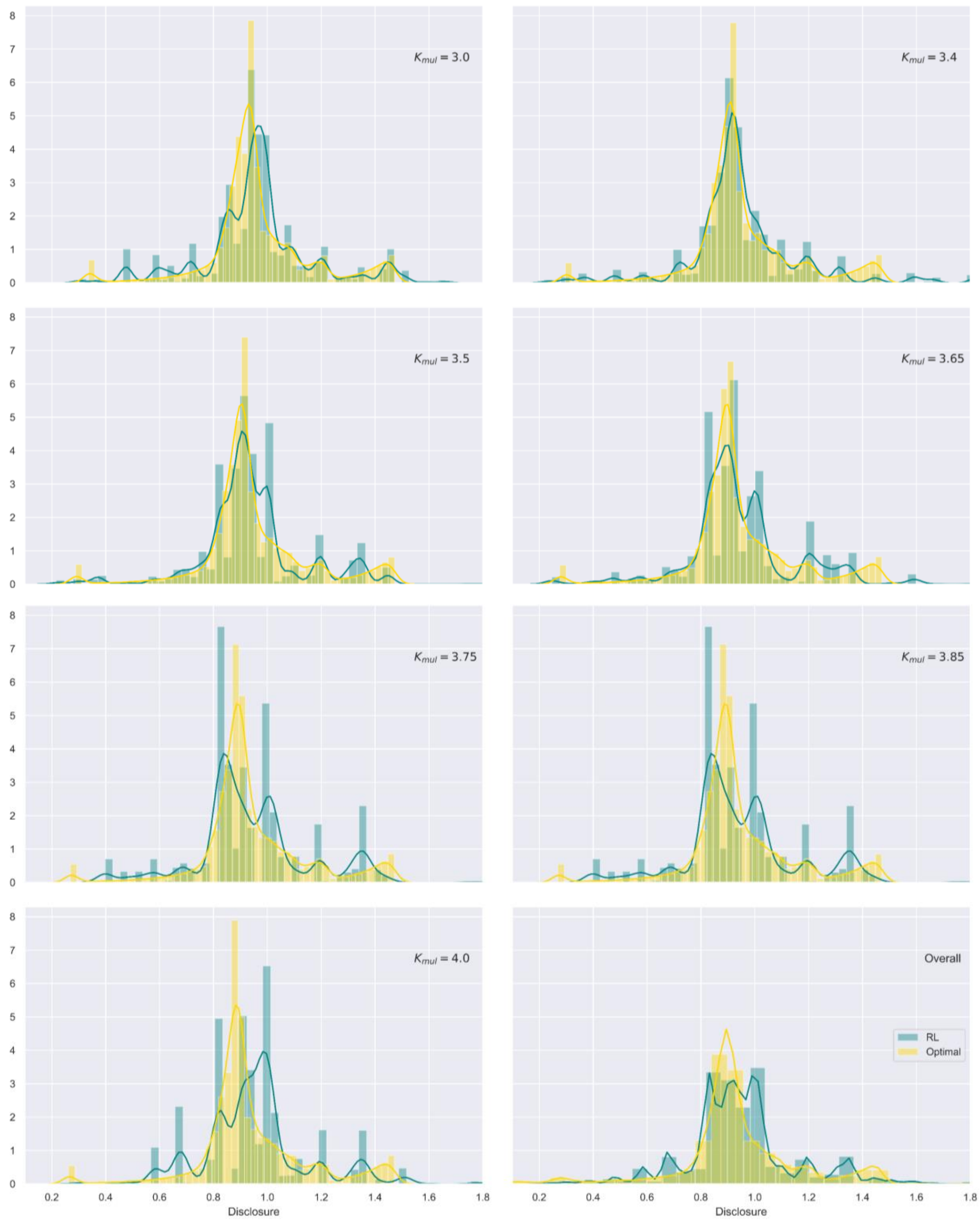

Figure 10 - The disclosure's histogram as a function of the multiplier, for the reinforcement learning and dynamic programming (optimal) policies.

Figure 10 depicts the disclosure histogram as a function of the applicable multiplier, considering each manipulative strategy. Generically speaking, the policies' histograms seems to be similar, exhibiting heavier tails on the right side as a function of the increasing multiplier. Such attribute is compliant with a risk-averse agent, which tends to report higher values to avoid further exceedances, especially in the penultimate multiplier – which corresponds to 4.0 – where an additional exceedance, would lead to a very negative state. However, whereas the optimal policy's histogram seems to be concentrated around a given mean, for each of the individual multipliers, the AI policy exhibits multiple and frequent peaks, leading to a distribution which, unlike the optimal policy, does not resemble a bell-shaped curve in most multipliers. In fact, the RL policy's histogram seems to be characterized by a certain degree of skewness, right-skewness for multipliers 3.5 to 3.85, and the left-skewness in the last multiplier, 4. These features can be summarized by observing the last histogram, which depict the policies' population. Notice that the optimal policy exhibits a fairly smoothed and well-defined bell shape around 0.90% whilst the estimated policy is characterized by a plateau between 0.80% and 1.05%.

Before proceeding to the Monte Carlo benchmarks, in which the actual return of the estimated policy can be assessed, a small simulation is introduced, which aims at establishing how conservative the estimated policy is, regarding its optimal counterpart. In order to do so, the policy has been injected into the simulation environment and the final episodic state recorded, that is, the multiplier's variation on each complete backtesting period. This trial ran 1,000,000 times.

Initial K	Reinforcement Learning		Dynamic Programming	
	Final K		Final K	
	Increased	Decreased	Increased	Decreased
3	81.92%	18.68%[10]	93.63%	6.37%[10]
3.4	67.93%	32.07%	87.54%	12.46%
3.5	42.99%	57.01%	74.74%	25.26%
3.65	19.25%	80.75%	57.14%	42.86%
3.75	3.99%	96.01%	32.76%	67.27%
3.85	0.13%	99.87%	4.06%	95.94%
4	N/A	100.00%	N/A	100.00%
Average	30.83%	69.17%	57.22%	42.78%

Table 5 - Multiplier evolution considering the reinforcement learning policy under an environment simulated 1.000.000 times.

The simulation results point towards the reinforcement learning policy being more conservative and less risk prone than its dynamic programming counterpart. Not only did the estimated

[10] For the specific case of the default multiplier, 3, the value refers to percentage of times the multiplier remained equal, not decreased.

policy record far higher decrease rates – nearly doubling in at least three multipliers -, but the policy avoided increasing the multiplier at the end of the period, decreasing or maintaining it in 69.17% of the cases on average, against Seixas (2016)'s optimal policy, which recorded a 42.78% multiplier rise rate. Both policies recorded a 0.00% bankruptcy rate. This conservative behavior comes at the cost of higher disclosure values, hence smaller savings. This behavior is contradictory to that which is expected regarding table 4, considering the RL policy's lower disclosure mean. However, recall that table 4 represents the statistical indicators for the entire policy hence, not valid for justifying the individual behavior each for each trajectory.

The brief analysis performed throughout the current paragraph is consistent with that of an agent which has learnt the contours of the environment and the optimization objective at hands. With this said, the following paragraph compares the performance of the RL policy under a Monte Carlo simulation, against its counterpart, the optimal policy, and the alternative, the normal distribution policy.

4.2.4 Policy Benchmarking

The previous section discussed to what extent the reinforcement policy converged to its dynamic programming counterpart relying on the use of key statistical indicators. However, the object of interest is to measure the estimated policy's performance, which, in Finance, corresponds to portfolio returns.

The current section presents an in-depth comparison between the results obtained by Seixas (2016), the normal or non-manipulative disclosure rule, and the introduced RL methodology, via Monte Carlo simulation. Under such framework, the daily returns were set to follow a normal distribution with mean zero and standard deviation equal to 1.7%, in accordance with Seixas (2016)'s. Accordingly, the simulation's return for a given time-step t, is given by the inverse of a random number sampled from a univariate Gaussian distribution with mean 0 and variance 1

$$r_t = \phi^{-1}(rnd_t) \tag{42}$$

$$rnd_t \sim N(\mu, \sigma^2)\,, \tag{43}$$

where r_t represents the period's random return, constrained to the interval [-1, 1], and μ and σ, represent the random variable's distribution mean and variance, corresponding to 0 and 1, respectively.

An introductory note is required, as a key difference between the two author's simulation arises. Whereas Seixas (2016)'s average of the last sixty disclosed VaRs considered a standard, unmanipulated value for non-existing days – that is, when the simulation process initiates there are no records of its past, in such cases, the preceding author considered such values to equal 1 –, the present work considered the running mean of the existing disclosures. This seemingly simple difference renders direct per author result comparison futile, at least, on the first simulations where the different line of thought's effects still endure. Hence, and because there is no telling when such effect will cease, the author opted for simulating all three scenarios at once, not relying in Seixas (2016)'s results.

As mentioned in the beginning of the current chapter, the Monte Carlo simulation will employ three distinct policies: the introduced reinforcement learning policy; the optimal policy as obtained by Seixas (2016); and lastly, the normal policy, which corresponds to disclosing the computed value-at-risk, without manipulating its value. The constraints enforced in the RL environment during the training process, chapter 4.1, are applicable for the present simulation. Specifically, a maximum of 11 exceedances has been imposed, under the assumption that the agent would avoid such state at all cost, after all, the FI's purpose is to maintain its internal model. Violating the internal construct more often than foreseen in the defined multipliers, would result in the ECB enforcing a standard model instead. Accordingly, the simulation environment prevents the agent from reaching the bankruptcy state, by forcefully reporting the highest possible value – 300% of the value-at-risk – when the current exceedance number equals to 10.

Lastly, similarly to the RL environment, the simulation includes the possibility of bankruptcy, defined by a daily MRC smaller than the day's loss

$$r_t < MRC_t \, , \tag{44}$$

where r_t corresponds to the simulation return for day t, as defined in (42), and MRC_t corresponds to the market risk charge, as defined in equations (2) and (3).

Another important aspect is to deflect the impact of handling unobserved samples in the computation of the average of the last sixty disclosed values on the simulation. With this in mind, the first 10 simulations have been disregarded in an attempt to decrease the impact the rolling sample decision has on the analysis.

Lastly, it is important to note that each simulation starts at the default multiplier, 3 after which, the multiplier applicable for the following year is solely dictated by the simulation's performance.

The methodology considered a total of 1,010,000 simulations, where each simulation simulates 30 years.

Figure 11 depicts both simulation's exceedance's relative and cumulative frequencies, as a function of the employed policy. The policies exhibit a similar behavior regarding reporting violations throughout the simulation, despite the reinforcement learning approach, having recorded lower frequencies of exceedances up to and including the 4 exceedance threshold – rendered in opaque orange –, resulting in the optimal policy yielding a higher frequency of minimum multiplier selection. Beyond the mentioned limit, the reinforcement learning simulation recorded significantly higher exceedance frequencies in the 5 to 8 range. This trend is reversed at the 9th level, where the optimal policy records slightly higher frequencies than its counterpart.

The cumulative frequency curves, depicted on the right axis of each figure, assist in clarifying the overall behavior of each policy. Despite its similar behavior, the optimal policy recorded a cumulative frequency around 67% in the 4th exceedance threshold, whereas its counterpart only does so at around 60% of its simulations.

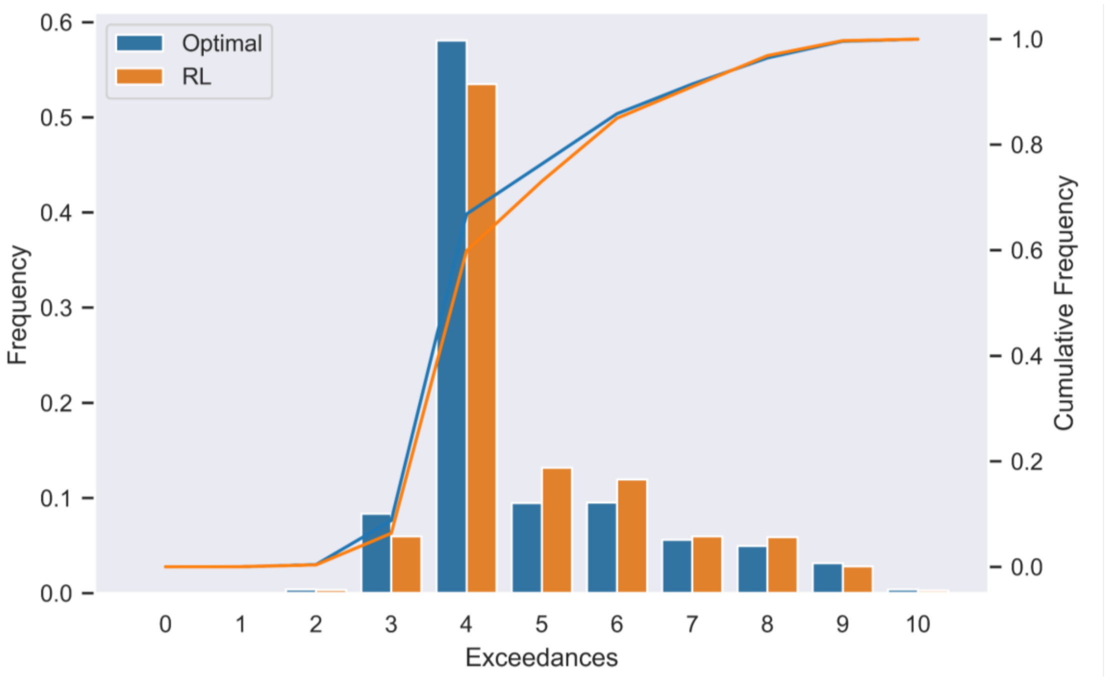

Figure 11 - Exceedance relative frequencies, for both RL and Optimal Policies, under the simulated Monte Carlo environment. The continuous lines represent the cumulative frequency of each property, measured on the secondary axis.

Shifting the focus onto the normal policy, its exceedance histogram seems to be concentrated around 2 violations, with a positive skew around its median, 2. Such behavior points towards the normal policy focusing on the lower tail of multipliers – until the 4th exceedance level – in around 89% of the simulations, according to its cumulative relative frequency curve, an expected behavior considering its non-manipulative behavior. The divergence of results in terms of outcomes is very clear, for the reinforcement learning strategy results in minimum multiplier selection, at around 60% of the times, which will be visible in figure 13.

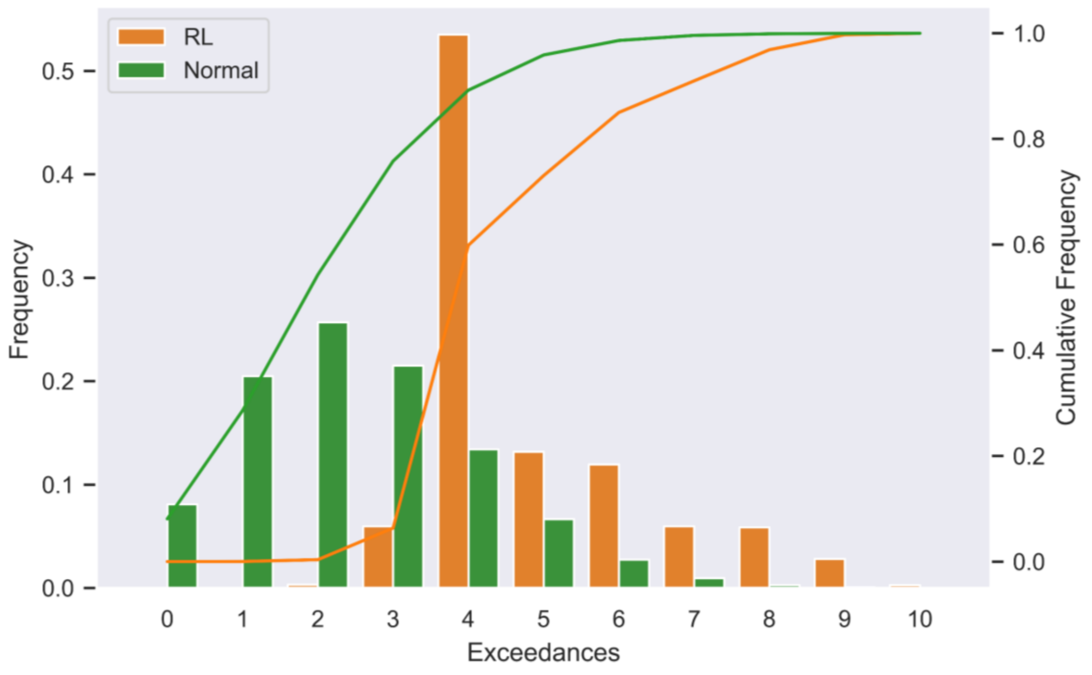

Figure 12 - Exceedance relative frequencies, for both RL and Normal Policies, under the simulated Monte Carlo environment. The continuous lines represent the cumulative frequency of each property, measured on the secondary axis.

Strategy	Mean	Median	Mode	Max	Min	σ	IQR	Q_1	Q_3
RL	4.87	4	4	10	0	1.48	2	4	6
DP	4.74	4.74	4	10	0	1.50	1	4	5
Normal	2.5	2	2	10	0	1.57	2	1	3

Table 6 - Exceedance descriptive statistics under the Monte Carlo simulation.

The multiplier histogram is a direct reflection of the exceedances' frequency distribution, as the latter dictates the former, according to table 1. Consequently, and according to the analysis of figure 11, the reinforcement learning strategy yields a default multiplier around 7% less frequently than its optimal counterpart. As for the 1 to 2 multiplier index range, the reinforcement learning policy records significantly higher relative frequencies, for a maximum difference of around 3% in the second multiplier – represented by the index number 1. Considering the remainder of the histogram, the RL strategy achieves higher relative frequencies for multiplier indexes up to and including 4. However, the tendency is reverted after the fourth multiplier, as the optimal policy yields a higher frequency of fifth multiplier index selection, and even reaching the last multiplier, which its counterpart does not do so.

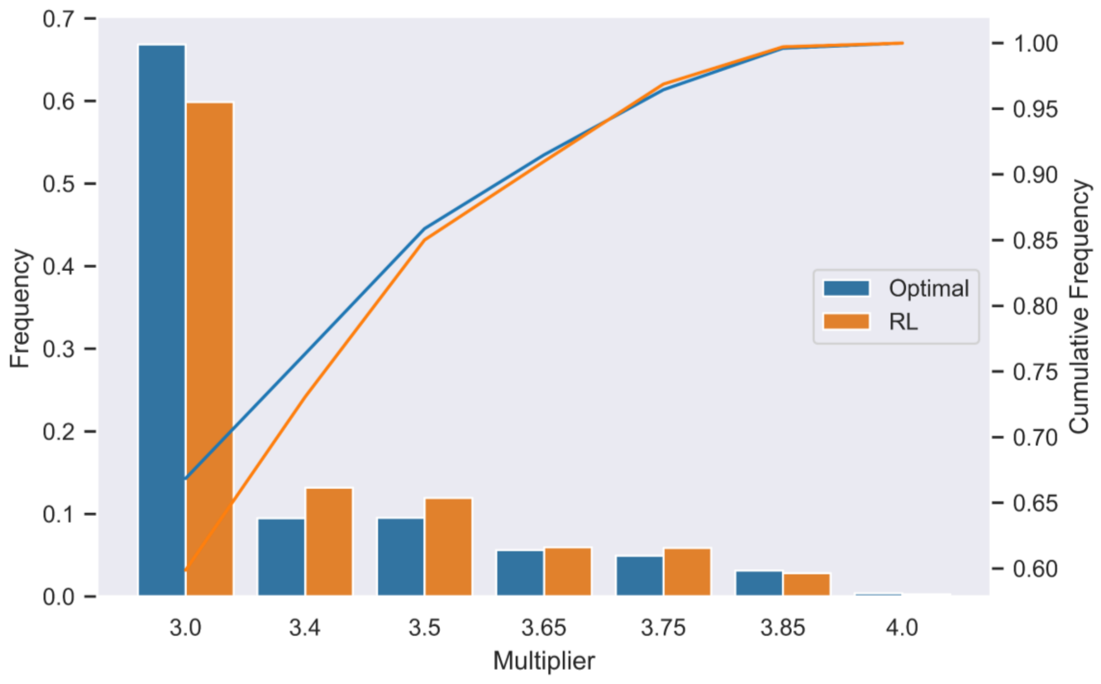

Figure 13 - Multiplier index's relative frequencies, for both RL and Optimal Policies, under the simulated Monte Carlo environment. The continuous lines represent the cumulative frequency of each property, measured on the secondary axis.

Figure 14 compares the reinforcement learning and normal policies' multiplier selection histogram. As expected, the normal simulation recorded the default multiplier in most simulations, specifically, 89% of these, decreasing the per-multiplier frequency in a seemingly log-normal form from that point forward, up until the third multiplier index. The figure highlights the disparity between the two policies, with the artificial intelligence rule achieving the same cumulative frequency as its non-manipulative counterpart's default multiplier frequency, at the third multiplier index.

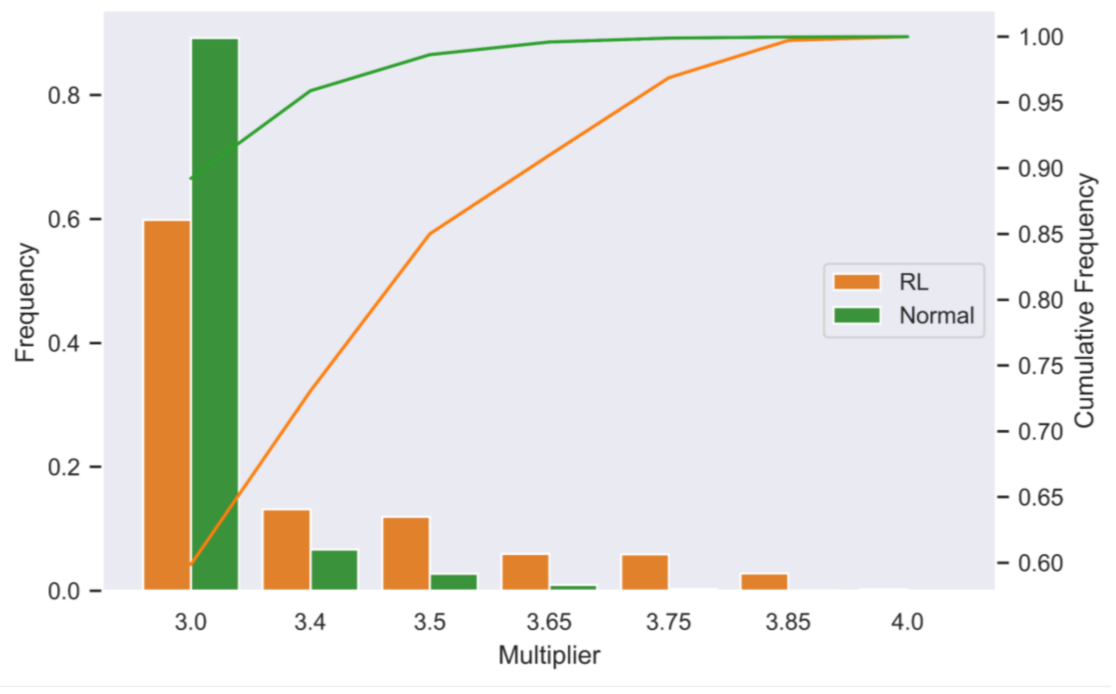

Figure 14 - Multiplier index's relative frequencies, for both RL and Normal Policies, under the simulated Monte Carlo environment. The continuous lines represent the cumulative frequency of each property, measured on the secondary axis.

Strategy	mean	median	mode	max	min	σ	IQR	Q_1	Q_3
RL	0.94	0	0	6	0	1.41	2	0	2
DP	0.83	0	0	6	0	1.42	1	0	1
Normal	0.17	0	0	6	0	0.22	0	0	0

Table 7 - Multiplier descriptive statistics under the Monte Carlo simulation.

Figure 15 represents a side-by-side plot of the RL and DP policies' market risk charge histogram, respectively, on the left and right side of the figure. The AI policy's histogram revolves around 0.37, close to the DP solution's mean, 0.36. The histograms appear to exhibit a degree of positive skewness, however, this statement is only true for the optimal policy, whose descriptive statistics respect the condition $mode < median < mean$. Nonetheless, whereas the left histogram depicts a relatively continuous plot, the right-side figure seems to be composed of two different bell-shaped distributions. The data is clustered under the same range, 0.30 to 0.45 for both policies.

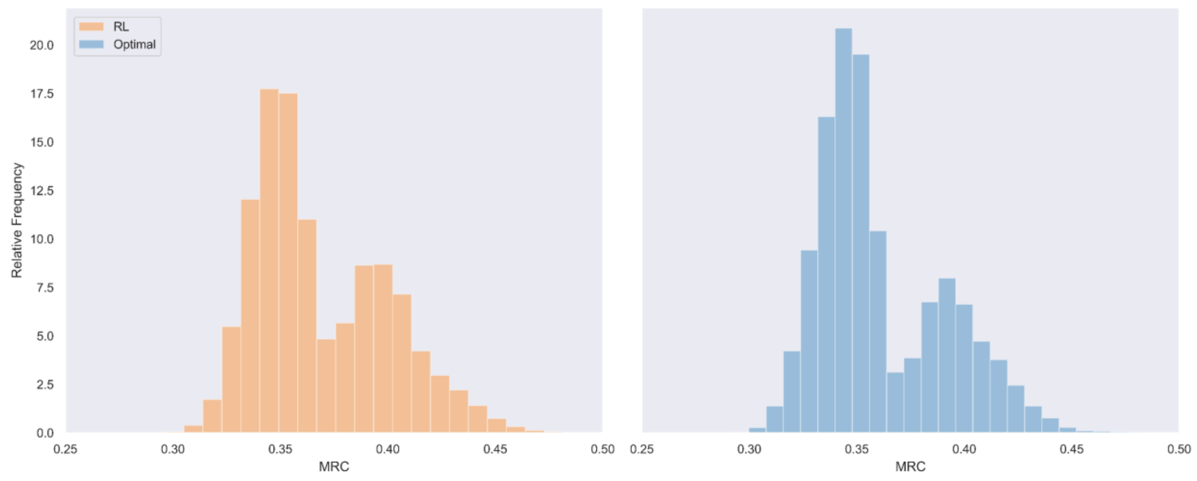

Figure 15 - Disclosure relative frequencies under the optimal and reinforcement learning policies, for the simulated Monte Carlo environment.

The normal policy's MRC distribution, the right-hand side of figure 16, is characterized by a large disclosure bin, corresponding to a fixed value of 0.38. This is an expected behavior, after all, the policy reports the computed value-at-risk amount without adjusting its value, which results in its multiplier being constant throughout the majority of the simulations. With this said, its disclosure histogram is expected to be concentrated in the value corresponding to a default multiplier unmanipulated disclosure, 0.38.

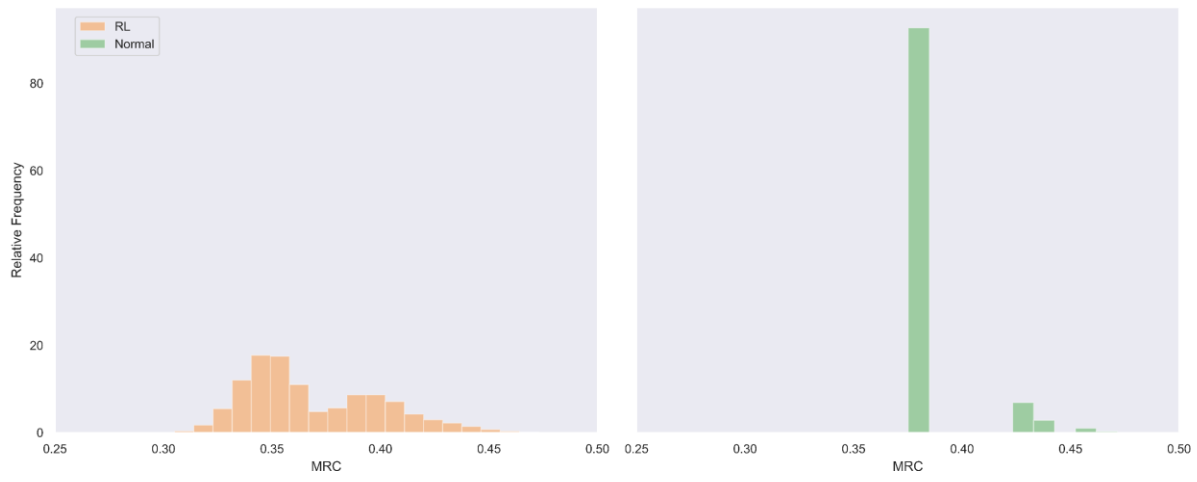

Figure 16 - Disclosure relative frequencies under the Normal Policy, for the simulated Monte Carlo environment.

Strategy	Mean	Median	Mode	Max	Min	σ	IQR	Q_1	Q_3
RL	0.37	0.36	0.37	1.01	0.28	0.03	0.05	0.34	0.39
DP	0.36	0.35	0.32	0.69	0.26	0.03	0.05	0.34	0.39
Normal	0.38	0.38	0.38	0.85	0.38	0.01	0	0	0

Table 8 - MRC descriptive statistics under the Monte Carlo simulation.

The last point of interest is to evaluate the additional returns the institution may obtain from using the introduced policy, as a result of invested the capital made available from the smaller disclosed value. Regarding financial savings, these have been computed by assuming the institution desires to maintain their exposure, that is, to maintain its risk assessment constant, hence the daily investment is constrained to ensure a fixed daily MRC, allowing the invested amount to vary according to the resulting risk charge. In other words

$$Investment_t * Price_t * MRC_t = MRC_{fixed} \, , \tag{45}$$

where $Investment_t$ represents the capital invested at time step t, $Price_t$ corresponds to the asset's price for the given time period, and MRC_t and MRC_{fixed} correspond to the market risk charge computed at time step t and the target or fixed value for the MRC, respectively.

Strategy	Mean	Median	Max	Min	σ
RL - DP	-0.53%	-1.35%	2.63%	-5.74%	0.90%
RL - Normal	4.32%	4.36%	9.85%	-1.29%	1.16%

Table 9 – The average relative annual gain descriptive statistics under the Monte Carlo simulation.

Table 9 measures the reinforcement learning approach's increased average annual return, in comparison with their optimal and non-manipulative counterparts. As expected, the artificial intelligence strategy underperforms regarding the dynamic programming approach but results in considerately larger returns in comparison with the normal policy. In fact, the RL policy generated, on average, an average annual return only 0.53% below its optimal counterpart in some cases, managing to overperform in 2.63%, which can be explained by the policy's large variation.

4.2.5 Model Limitations

As described by Campbell (2005), one of the conditions to classify a VaR model as adequate, is the Unconditional Coverage Property. This hypothesis states that the likelihood of a loss higher than the disclosed VaR occurring, is exactly equal to $\alpha * 100\%$, where α is the value-at-risk significance level

$$P[I_{t+1}(\alpha) = 1] = E[I_{t+1}(\alpha)] = \alpha \, , \tag{46}$$

where $I_t(\alpha)$ represents the hit function, an indicator function which takes the value one if an exceedance is recorded, and zero otherwise

$$I_{t+1}(\alpha) = \begin{cases} 1 \, if \; x_{t,t+1} \leq -VaR_t(\alpha) \\ 0 \, if \; x_{t,t+1} > -VaR_t(\alpha) \end{cases} , \tag{47}$$

As referred by Campbell (2016), a problem arises when the value-at-risk estimation model consistently under or over estimates the actual risk amount, which is a critical assumption of the present work. Unconditional coverage tests tend to be inadequate when it comes to detecting such situations, as for the sample size dictated by the regulatory framework, the tests seem to showcase low power (Campbell, 2005). For this reason, disclosing manipulated values of the FI's risk may invalidate the model when it comes to unconditional coverage backtesting tests, such as Kupiec's proportion-of-failure (POF). Despite not being present in the current regulatory framework, this section should be looked upon as a disclaimer, should future versions of the Basel Accords include unconditional coverage tests, research based on disclosure manipulation might no longer be valid. Additionally, for FI's seeking to integrate this research onto their risk estimation framework, even if not required by the Basel regulation, special attention should be taken upon assessing the quality of the models as again, unconditional coverage-based tests, such as the POF assessment of the value-at-risk model's unconditional coverage property, may not be applicable.

4.2.6 Policy Gradient Advantages

Throughout the course of chapters 1 and 4, numerous references to the fact that reinforcement learning models do not require an explicit transition model have been made. However, no further exploration on the consequences and benefits of such capability have been explored.

The current section aims at identifying some of the key advantages deep reinforcement learning has over dynamic programming.

Starting with the ability to operate and learn under an environment whose transitions are unknown. Environments created under a controlled setting, as is often the case of research papers, might not pose a major obstacle in terms of modeling the transition probabilities, thus being ideal candidates for dynamic programming, often at the expense of simplification, which results in inaccurate results. Furthermore, under such simplifications, the environments are often disconnected from reality, rendering the applicability of such findings unfeasible. However, man-made environment models which attempt to operate in real world situations, cannot fathom to capture the minutia and complexity of the underlying dynamics, often not describable through known mathematical formulas – such as the case of financial markets, whose complexity lies beyond the current theoretical knowledge. This means that deep reinforcement learning models, policy gradient based in particular, serve as prime candidates for such cases, as they are able to estimate an optimal policy despite not knowing nor having direct access to the environment’s transition probability model. Combined with the ability to estimate both discrete and stochastic policies, and to learn discrete and continuous action and state spaces, DRL arrives with the promise of unbounded and unprecedented learning in financial markets. In fact, several applications and research have proven successful, in using AI for algorithmic trading (Liang, et al., 2018) and (Huang, 2019), portfolio management (Park, et al., 2019) and (Wang, et al., 2019), among others.

Deep Reinforcement Learning models don’t require modifications when the underlying environment, including its transition probabilities, are changed. Although more hyperparameter tuning will likely be needed, including modifying the neural network’s width (number of neurons per layer) and or depth (number of layers), the model will still operate, and learn. Note however, that for the sake of transparency, should the actions space be of a discrete nature, modifying the number of states will require architectural modifications. This translates not only in the ability to vary the environment’s architecture and complexity whilst retaining the model, but also in the possibility of using different environments - for instance, one for low volatility periods, and another for high volatility periods - during the same training process.

Another key advantage is the ability to not only learn under continuous action and state spaces, but also to learn both discrete and continuous policies. The use of dynamic programming translates in discretized action and state spaces, often due to hardware (RAM) limitations.

Doing so, reflects in a compromise between results and non-captured model nuances, as under such scenario, computation resources and model accuracy are used interchangeably. Policy Gradient on the other hand, can handle both discrete and continuous spaces, whilst relying on the use of CPUs and GPUs – as neural network operations, the model's core, are comprised of matrix operations – to solve the optimization problem.

Finally, deep reinforcement learning models can be frozen and re-started by saving the function approximators' parameters θ. Such results in models which can be updated posteriorly, if new data is available, allowing resources to be saved, and the learning process and data ingestion to take place simultaneously. Often samples are not readily available, as is the case of financial market data, which is nonexistent overnight for domestic markets. In addition, static parameters open the possibility of interchangeable models, often used in transfer learning, drastically decreasing the cost and execution time of training variations of the base model.

The presented aspects are critical regarding optimization in Finance, where the environments are known to be hard to model, continuously changing, consisting of infinite action and state spaces.

4.2.7 Deep Reinforcement Learning Limitations

Reinforcement learning, the model-free path in particular, in its current state of affairs, is not the holy grail of MDP solving, in fact, it suffers from a variety of problems which severely hinder the learning process, in both accuracy and speed. The algorithms, especially the on-policy cases, are extremely sample inefficient, as they lack an environment model, thus using the samples to indirectly approximate said construct through bootstrapping, whilst at the same time, relying on neural networks, a framework which is, by itself, sample inefficient due to the slow pace of gradient descent. Like dynamic programming, this class of algorithms suffers greatly with the so-called curse of dimensionality on large state and or action spaces, however, not in terms of memory resources – RAM – like DP, but in sheer computational power and approximation complexity. Deep Reinforcement Learning algorithms suffer from a typical problem derived from the use of neural networks, and that is the fact that they are opaque, in the sense that they lack both predictability and explainability, constituting what is typically referred to as black box models. DRL models are narrow, as they lack generalization capabilities, which in essence, signifies that transferring knowledge from one environment to the next, similar as they may be, is brittle and often unachievable, caused by overspecialization in the first environment. Unlike value-based methods, policy gradients, tend to be stuck in local minima, whilst taking long to converge. Lastly, the environment's rewards are not a straightforward implementation, as any bad design will be thoroughly exploited by the algorithm, known as reward hacking.

Whilst the first problem can be improved through the use of model-based RL algorithms when some of the environment's dynamics are reproduceable, which are more sample efficient, they still lack the capability of properly tackling the other mentioned problems. In addition, the use of model-based RL generates its own set of problems, namely, model estimation risk.

The presented arguments against the use of deep reinforcement learning don't signify that said frameworks are not to be used. The purpose of the present chapter is to reinforce the idea that the right tools should be selected for the problem at hands. Deep Reinforcement Learning opens the door towards solving problems which were simply intractable in the past. However, when the environment's transitions are known, or approximately known – as is the case in Partially Observable Markov Decision Processes –, traditional solutions are often a faster, more robust, cheaper and better result producing options.

4.2.8 Sub-optimal Optimality

As referred in section 4.2.3, the RL policy exhibits a higher variance than Seixas (2016)'s, and is likely to do so under the deep reinforcement learning framework, given its generalizing nature. However, this behavior might be desirable. Unlike the optimal policy generated by dynamic programming, its sub-optimal counterpart does not follow a strictly increasing trend, which is an expected behavior of financial market risk measures. Thus, following the proposed policy might help cover the FI's tracks by obfuscating its disclosure manipulation, a behavior which the ECB might frown upon. In a similar line of though, increasing the value-at-risk disclosure by nearly threefold in a question of fifty trading days – for instance, considering the optimal policy for the default multiplier – would likely raise some concerns at the supervisory authorities, as it could flag the model as being too variance-sensitive, in spite the inclusion of the average of the last sixty disclosed value-at-risk values on the MRC.

5 Conclusion

The aim of this work was to demonstrate the adequacy and capabilities of deep reinforcement learning in solving financial optimization problems characterized by a complex Markov decision process. For this purpose, a policy generated by DRL on the VaR disclosure optimization problem has been benchmarked against an optimal policy created by Seixas (2016) via dynamic programming.

The proximal policy optimization agent's policy showcased strong signs of convergence with its optimal counterpart, despite not being given any formal knowledge of the environment's dynamics, and the fact that the selected algorithm is not the optimal choice regarding discrete spaces, as described in section 4.2.1. As explored in section 4.2.3, the yielded policy converged, on average, to the optimal policy. However, an in-depth analysis revealed the presence of high variance in the estimated policy, partly due to the small number of iterations, and a tendency to under-report regarding its optimal version, that is, the former tends to disclose smaller VaRs than the latter. This behavior can be attributed to neural network's predisposition to generalize, being prone to report more stable values, and again, to the low number of learning iterations, which translates in the estimated state value function q_π not being able to perceive the true cost associated with a given state-action pair, hence, not matching the optimal state-value function q_*. Nonetheless, the policy generated by the artificial intelligence approach managed to obtain an additional average return only 0.53% lower than its optimal counterpart, which corresponds to the freed-up capital being invested at a fixed rate of 6%, contingent on the financial institution's MRC remaining constant.

Having demonstrated the algorithm's capabilities in the discrete space, the path is paved for its application in the more complex environment, thus removing the simplifications and limitations introduced to make the problem tractable via dynamic programming. Specifically, the introduction of continuous action and state spaces, allowing for the direct optimization of the Capital Risk Charge equation, as well as the optimization of multiple variables, including the stressed value at risk. Additionally, and perhaps more important, the need to assume a transition distribution is eliminated, as well as the rules which define it.

In addition to the advantages mentioned above, the use of PPO is able to generate solutions to the problem in question which, despite being sub-optimal, might be desirable. As discussed in section 4.2.8, the optimal policy is prone to raising concerns on the regulatory authority, the

European Central Bank, given the that the DP policy, although mathematically optimal, does not meet the expected behavior in financial markets (mean-reversal exhibiting a certain degree of variance, although one might argue that the variance should originate in the portfolio itself, hence, on its VaR, not on its disclosure policy), being characterized by a linear growth, often triplicating its disclosed value in merely twenty days, whose trend and fixed nature is easily perceived. DRL on the other hand, is capable of generating stable policies around a given value – due to neural network's ability and propensity to generalize – thus promoting, by default, solutions that are stable in the long-run, which can simultaneously maximize the short-run variance, resorting to techniques such as the ones suggested in chapter 5.

On a final note, it is important to be aware that the application of DRL is both frail and expensive, in both time and financial capital, being the subject of extensive experimentation and optimization, relying extensively on the researcher's experience and domain-knowledge. For these reasons, and according to the results presented in this thesis, the use of deep reinforcement learning is advisable solely in cases in which dynamic programming proves inappropriate, even if at the expense of a few simplifications.

6 Future Work

One of the key-aspects regarding possible improvements to the exhibited work, lies on the employed algorithms. Firstly, as pointed in section 4.2.6, PPO and similar algorithms suffer from a variety of problems. One way to overcome them, might be through the use of hierarchical reinforcement learning, or meta reinforcement learning, new approaches which seem to be gaining ground on the control aspect reinforcement learning.

Another possible line for improvement lies on extending the model defined in 4.1 to allow for an approximation to the complete problem which FI's face on a daily basis, whilst at the same time, providing a framework which has the ability for continuous adaptation to the institution's portfolio, given DRL's ability to estimate continuous action and state spaces, whilst also being capable of being continuously improved with new market samples. The present section aims at introducing a suggested environment which closely mimics equation (3).

The improved model's variables would be the same as those present in the base model. The natural next-step would be to add a record of the last sixty VaRs selected within the current episode, onto the environment. Such variable could be named VaR_{60}, and it would be expressed by an array, being later on, merged into the reward function, hence, not adding further complexity to the model.

$$VaR_{60} = \{var_{t-60}, var_{t-59}, \dots, var_{t-1}\}\,, \tag{48}$$

The previous addition does have one pitfall. One of the key-assumptions of MDP's is *memorylessness*, meaning, each Markov state contains all required information from the state's history. For this reason, keeping track of previously selected actions via an external variable, could prevent convergence as one of the assumptions would be violated. Evidently, doing so would force the algorithm to optimize without ever attaining full knowledge of the model's constraints, as parts of it would be invisible. For this reason, the inclusion of such history would have to be performed via injection into the observation, as required by MDPs. In practice, this would lead to the creation of a new observation subspace, specifically

$$\mathcal{V} = \{x \mid x \in \mathbb{R}, 0 \leq x \leq 3\}\,, \tag{49}$$

thus modifying the observation space (33) to

$$observation_t = (\mathcal{T}, \mathcal{E}, \mathcal{K}, \mathcal{V})\,, \tag{50}$$

Under such modification, the observation space would no longer fully discrete, as it contains a continuous component, induced by the inclusion of a variable from subspace $\mathcal{V}$.

At the same time, the discretization of the action space could be removed, modifying (34) to a continuous interval

$$\mathcal{A} = \{n \mid n \in R^+, n \leq 3000\}\,, \tag{51}$$

Another obvious problem arises with this solution: if each episode is independent and thus, the selected actions (or VaRs) only reflect decisions made on the current episode, then except for when the elapsed time or visits is equal or larger than sixty, there would not be a true recollection of the past sixty actions, given no such information existed.

As pointed in section 4.2.8, a policy suitable for financial markets is bound to showcase a certain degree of volatility, so as to mimic the underlying market conditions. With this in mind, the reward function could be expanded, penalizing a lack of variation between subsequent actions, in other words, maximizing long-term action variance. However, excessive short-term variability is also undesirable, as it would generate an unstable policy. Hence, the reward function should promote long-term variance, whilst maximizing short-term action similarity. The former is already present in PPO via the entropy bonus, the latter on the other hand, could be achieved through the use of regularization, for instance, temporal regularization (Thodoroff, et al., 2018), in the algorithm's objective function. The main idea of said work, is to enforce regularization in the actions along the trajectory, penalizing significant changes between subsequent actions, instructing to algorithm to promote short-term decision similarity. Considering proximal policy optimization relies on the use of the advantage estimator $\hat{A}_t$ introduced in equation () of Appendix A.III – Policy Gradient, and rewriting said function as

$$\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_T\,, \tag{52}$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)\,, \tag{53}$$

as defined by Thodoroff et al. (2018), the regularization's objective being given by the term

$$\delta_t^{\beta} = r_t + \gamma\big((1-\beta)v(s_{t+1}) + \beta\tilde{v}(s_{t-1})\big) - v(s_t)\,, \tag{54}$$

with $\tilde{v}(s_t)$ corresponding to exponential smoothing given by

$$\tilde{v}(s_t) = (1-\lambda)v(s_t) + \lambda\tilde{v}(s_{t-1})\,, \tag{55}$$

introduced in the Critic's objective function, with β and λ representing the regularization terms.

Due to both financial and time restrictions, such experimentation is left for future work.

Besides the algorithmic aspect, it is important to insist that this study does lie behind the current legislation. Specifically, our study is a direct applicability of the Basel II accords, whereas at this point, Basel 2.5 is imposed, with ECB having already begun the process of implementing the third Basel accord, set to be enforced from January 1st of 2022 onwards. The inadequacy arises from the fact that the present work does not contemplate the concept of stressed value-at-risk (sVaR), a major player in the Basel 2.5, let alone in its successor. This drawback renders a direct applicability of the achieved optimal policy useless, or at least, ineffective. Thus, future developments on the described research should focus on at least, the integration of the stressed value-at-risk and its disclosure in the refined environment.

7 Bibliography

Agarap, F. M. A., 2019. *Deep Learning using Rectified Linear Units (ReLU)*.

Alexander, C., 2008. *Value-at-Risk Models.* John Wiley & Sons, Ltd.

Allen, B., Chan, K. K. & Thomas, S., 2012. Basel III: Is the cure worse than the disease? *International Review of Financial Analysis,* 25(C), pp. 159-166.

Artzner, P., Delbaen, F., Eber, J. & Heath, D., 1999. Coherent Measures of Risk. *Mathematical Finance,* 9(3), pp. 203-228.

Basel Committee on Banking Supervision, 1996. Supervisory Framework for the use of "Backtesting" in conjunction with the Internal Models Approach to Market Risk Capital Requirements.

Beyerer, J., Niggemann, O. & Kühnert, C., 2016. *Machine Learning for Cyber Physical Systems - Selected papers from the International Conference ML4CPS 2016.*

Campbell, S. D., 2005. A Review of Backtesting and Backtesting Procedures. *Finance and Economics Discussion Series*, Volume 21.

Danielsson, J. & Zigrand, J.-P., 2003. *On time-scaling of risk and the square–root–of–time rule.* London School of Economics .

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning.* MIT Press.

Huang, S., 31 Jul 2019. *Taxable Stock Trading with Deep Reinforcement Learning.* National University of Singapore.

Korb, K. B. & Nicholson, A. E., 2004. *Baysian Artificial Intelligence.* Chapman & Hall/CRC.

Kuo, C.-K., Lee2, C.-W. & Kuo, W., 2013. Forecasting Value at Risk: A Strategy to Minimize Daily Capital Costs. *ACRN Journal of Finance and Risk Perspectives,* 2(1), pp. 39-49.

Liang, Z. et al., 2018. *Adversarial Deep Reinforcement Learning in Portfolio Management.* Sun Yat-sen University.

McAleer, M., 2008. The Ten Commandments for Optimizing Value-at-Risk and Daily Capital Charges. *Journal of Economic Surveys,* 23(5), pp. 831-849.

McAleer, M., Jimenez-Martin, J.-A. & Pérez-Amaral, T., 2009. A Decision Rule to Minimize Daily Capital Charges in Forecasting Value-at-Risk. *Journal of Forecasting,* 29(7), pp. 617-634.

Miranda, M. J. & Fackler, P. L., 2002. *Applied Computational*.

Ogbechie, A., Díaz-Rozo, J., Larrañaga, P. & Bielza, C., n.d. *Dynamic Bayesian Network-Based Anomaly Detection for In-Process Visual Inspection of Laser Surface Heat Treatment.* Technical University of Madrid, School of Computer Science.

Park, H., Sim, M. K. & Choi, D. G., 18 Jul 2019. *An intelligent financial portfolio trading strategy using deep Q-learning.* Pohang University of Science and Technology & Kyunghee University.

Pérignon, C., Deng, Z. Y. & Wang, Z. J., 2007. Do Banks Overstate their Value-at-Risk.

Pérignon, C. & Smith, D. R., 2010. The level and quality of Value-at-Risk disclosure by commercial banks. *Journal of Banking & Finance,* Volume 34, p. 362–377.

Schulman, J. et al., 2017. *Proximal Policy Optimization Algorithms.*

Sebe, N. & Lew, S. M., 2004. *Computer Vision in Human-Computer Interaction.* LNCS 3058 ed., Springer.

Seixas, M. R. d. S., 2016. *Optimal Value-at-Risk Policy: A model for minimizing the daily capital charge.* ISCTE Business School, Finance Department.

Simons, K., 2000. The Use of Value at Risk by Institutional Investors. *New England Economic Review,* Issue November/December.

Singh, S., Jaakkola, T., Littman, M. L. & Szepesvári, C., 2000. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, March, 38(3), pp. 287-308.

Sutton, R. S. & Barto, A. G., 2018. *Reinforcement Learning: An Introduction.* 2nd ed. The MIT Press.

Thodoroff, P., Pineau, J., Durand, A. & Precup, D., 2018. *Temporal Regularization in Markov Decision Process.* Montréal: 32nd Conference on Neural Information Processing Systems (NIPS 2018).

Wang, J. et al., 2019. *AlphaStock: A Buying-Winners-and-Selling-Losers Investment Strategy using Interpretable Deep Reinforcement Attention Networks.* Beihang University, Tsinghua University.

8 Appendixes

Appendix A – Theoretical Foundations on Deep Reinforcement Learning

Appendix A.I – From Dynamic Programming to Reinforcement Learning

The present section delves into the core concepts behind reinforcement learning, to provide insights to those unfamiliar with the literature. It is an overview of the notions found in Sutton & Barto (2012), the original algorithm's publications, among others. The following chapters borrow heavily from the previous publications. Those familiar with the topic are encouraged to skip directly to chapter 4.

The term dynamic programming (DP) refers to a collection of algorithms whose purpose is to achieve an optimal policy under a Markov Decision Process (MDP).

The dynamic programming framework – known as Generalized Policy Iteration (GPI) – is comprised of two layers, policy evaluation, and policy improvement, which work together to help the algorithm converge onto an optimal policy.

Policy evaluation, also referred to as the prediction problem, allows computing the state-value function v_π for an arbitrary policy π:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t) = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] , \end{aligned} \tag{A.1}$$

Policy improvement on the other hand, assists in deciding whether a new policy π' yields better results than the current policy π, $\pi' \geq \pi$:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmin}}\, q_\pi(s, a) , \tag{A.2}$$

where $q_\pi(s, a)$ represents the action value function given by

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a)] , \tag{A.3}$$

The alternating cyclic combination of the two previous equations is what defines the GPI and guarantees strict improvement of one policy over its predecessor. Whilst the value function improves upon itself at each iteration so as to be consistent with the policy, the policy iteration function seeks converging itself to optimality, being greedy with respect to the value function.

$$\pi_0 \rightarrow^{evaluation} v_{\pi_0} \rightarrow^{improve} \pi_1 \rightarrow^{evaluation} v_{\pi_1} \rightarrow^{improve} \pi_2 \rightarrow^{evaluation} \dots \rightarrow^{improve} \pi_* \rightarrow^{evaluation} v^*$$

However, for many real-world scenarios, a perfect information MDP, that is, $\mathcal{P}_{ss'}^{a}$ and $\mathcal{R}_{ss'}^{a}$,is not available, which renders direct DP application unpractical, often resulting in excessive simplifications. Some classes of reinforcement learning algorithms overcome this limitation, thus being to obtain an optimal policy under an environment characterized by incomplete information.

Whilst not a part of the RL family, Monte Carlo (MC) methods allow learning solely from experience, hence not requiring an explicit transitions distribution, instead, they update their estimates using the sampled states, actions and rewards. These methods – not to be confused with the generic term, which refers to any estimation method based on random operations –, update their value function estimates and policy upon episode completion. This means that MC methods require G_t, averaging returns on an iterative manner.

When a model of the environment is not known, the focus shifts onto estimating state-action value functions. The reasoning behind this statement is straightforward, as when the transition's model is known, the optimization task consists of selecting for each state, which action leads to the optimal reward and desired states, analogous to the dynamic programming algorithm. However, when the aforementioned model is absent, it becomes necessary to attribute a quantifiable value for each action. For this reason, all algorithms presented hereinafter, present a computation of the q-function, q_π, as a part of its control problem[11] algorithm.

Monte Carlo's control method applies the GPI, just like DP, only whilst the latter's implementation computed the value function directly from the MDP, the former's learns the value function by experiencing the MDP's returns.

[11] The concept of a reinforcement learning algorithm's control version, refers to optimal policy approximation.

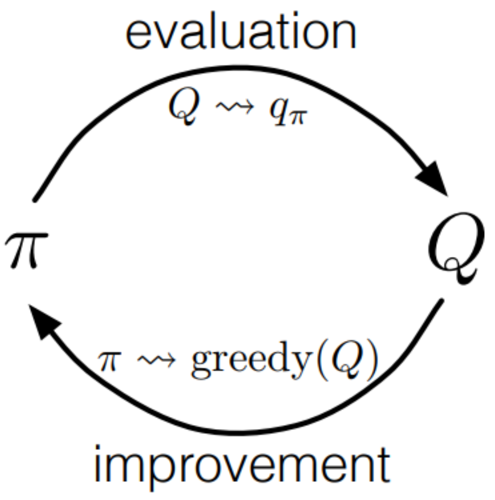
evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Figure 17 – The General Policy Iteration cycle for the Monte Carlo estimation for control.

The algorithm works iteratively focusing on three cyclic steps. The first step consists on generating samples for the episode according to the current policy π, relying on the use of greedy algorithms for simulated action selection, such as ϵ-greedy to balance between exploration and exploitation. Next, and assuming first-visit MC is used, the Q-value – recall that for the case of control methods, Q-value refers to the state-action pair's Q-value, capitalized given it is an estimate of the true q-value – is updated, as the average of the episode's returns

$$Q(S, A) \leftarrow \frac{1}{N} \sum_{t=0}^{N} G_t \, , \tag{A.4}$$

Finally, the third step improves the policy greedily with respect to the current value function:

$$\pi(s) = Q(s, a) \, , \tag{A.5}$$

Monte Carlo improves over dynamic programming by learning V_π and Q_π from direct environment interaction, not needing an explicit transitions model, and lastly, without the requirement of visiting all existing states. Nonetheless, the method does have faults of its own. MC RL requires episodic tasks – in opposition to continuous tasks –, it only updates its estimates from complete episodes, given the episode's return is only known after it ends.

Appendix A.II – Reinforcement Learning

Reinforcement learning refers to a set of algorithms aiming at learning how to behave in an environment whose dynamics are partially or fully unknown, by simple trial-and-error, where the reward may be perceived immediately, or posteriorly. These are the two core features of reinforcement learning (Sutton & Barto, 2018).

Sutton & Barto (2012) consider Temporal Differences (TD) Learning to be one of the core concepts of reinforcement learning. Unlike dynamic programming, TD learns directly from experience, not requiring a transition model. However, unlike Monte Carlo methods, TD does not require the episode's return G_t, instead, it updates its value function considering the next-time step's reward. This is referred to as bootstrapping.

At every subsequent time step, $t+1$, TD creates a target, the value function's value as estimated by the current step, and updates its previous step's estimate using the newly collected information:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]\,, \tag{A.6}$$

where R_{t+1} and $V(S_{t+1})$ represent the transitioned time step's observed reward and value function estimate, respectively, and $0 < \alpha \leq 1$ corresponds to a learning hyperparameter.

However, since TD(0) is a one-step algorithm, it would be unwise to update the function to match the new value estimate. For this reason, α was introduced, so as to point the update direction, without fully replacing it.

If one were to adapt the concept of estimation target to MC, the following equation would be returned

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]\,, \tag{A.7}$$

where the equation's right-hand side contains Monte Carlo's update target, $G_t - V(S_t)$.

Given the behavior described by () and (), TD overcomes some of MCs limitations, as it learns from continuing environments, is capable of learning online – after every step – and lastly, has the capacity to learn from incomplete episodes. Note however, that the algorithm referred to as TD, is a simplification of its actual representation, TD(0), where the integer 0 refers to the number of eligibility traces used by the algorithm, a concept beyond the scope of this introduction.

Reinforcement learning algorithms can be characterized by two sets of categories, which, broadly speaking, describe how the algorithm handles the underlying environment's model, and on the selection of samples for the policy target update. Regarding the former, these constructs can be either model-based, if they are provided a model of the environment or, if the algorithm learns it explicitly; or model-free, should they not depend on the existing model for the learning process to occur – as is the case for TD(0). As for sample selection, a RL algorithm can either be classified as being on-policy or off-policy. On-policy methods rely on states visited during the current trajectory to update its policy, whereas its off-policy counterpart uses a sample returned by any trajectory during the training process, or, in other words, the total discounted future reward is estimated under the assumption that a greedy policy has been followed, despite such not being true. Another way to look at this difference, is to consider on-policy methods evaluate or improve the policy which has been used to make decisions, whereas off-policy methods evaluate or improve any policy different from that which was used to generate the samples.

Applying TD(0) to the control problem yields two different algorithms according to the policy nature. Applied to an on-policy approach, yields the SARSA (State-Action-Reward-State-Action) algorithm. If, on the other hand, an off-policy view is employed, the resulting algorithm is named Q-learning.

SARSA earns its name due to the cyclic quintuple of events in a trajectory, on which the algorithm relies on to create its update rule.

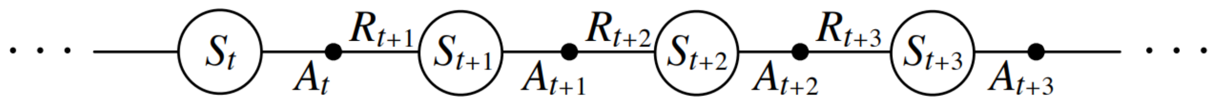

Figure 18 – SARSA trajectory illustration

Recall that for control algorithms, the focus shifts from state to state transition, and corresponding value learnt, to state-action to state-action pair transition, and value function approximation. In other words, from $v_\pi(s,a)$ to $q_\pi(s,a)$. Because SARSA is an on-policy algorithm, its update target employs the Q-value of the state-action pair transitioned to under the current policy π, continuously estimating q_π for the policy π, whilst modifying π with respect to q_π under greedy behavior.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]\,, \qquad \text{(A.8)}$$

The algorithm's update target is then defined by $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$.

```
Algorithm SARSA
1.   Input: step size $\alpha \in ]0,1]$, small $\epsilon > 0$
2.   Initialize $Q(s,a)$, for all $s \in S^{+}, a \in A(s)$ randomly, and set $Q(s_T,\cdot) = 0$
3.   for $k = 0,1,2,\ldots$ do
4.       Initialize $S$
5.       Choose $A'$ from $S'$ using a policy derived from $Q$ (e.g., $\epsilon$-greedy)
6.       for each step of the episode $t = 0,1,2,\ldots,T-1$ do
7.           Perform action $A$ observe $R$, $S'$
8.           Choose $A'$ from $S'$ using a policy derived from $Q$ (e.g., $\epsilon$-greedy)
9.           $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
10.          $S \leftarrow S'$; $A \leftarrow A'$
11.      end for – when $S$ is terminal
12.  end for
```

Algorithm 4 - SARSA pseudo-code.

Q-learning (Watkins & Dayan, 1992) is often classified as an early breakthrough for reinforcement learning, as the creation of an off-policy control algorithm under the TD framework.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)\,]\,, \qquad \text{(A.9)}$$

The previous equation immediately illustrates the off-policy component of Q-learning. Unlike its on-policy counterpart, SARSA, the algorithm directly approximates the optimal action-value function q^{*}, regardless of the underlying policy. However, the latter element still plays an important role in action selection, and consequently, state-action pair update. Under such construction, Q-learning's update target is given by $R_{t+1} + \gamma Q(S_{t+1}, a)$.

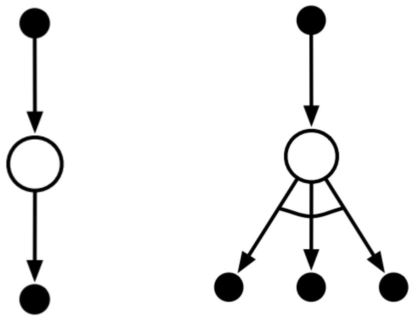
Figure 19 - The backup or target diagrams for SARSA and Q-learning, respectively.

Figure 19 illustrates the backup diagrams for SARSA and Q-learning. These figures conceptualize how the update target is defined. The root filled node represents an action node, as do the leaf nodes. The white circle represents a state-action node. SARSA performs a backup by updating the state-action node's q_π based on the action nodes transitioned from (root) and to (leaf). Conversely, Q-learning's state node update seeks maximizing over all possible actions available in the following state, where the arc connecting the multiple branches represents the maximum operator.

```
**Algorithm** Q-Learning
1.  Input: step size $\alpha \in ]0,1]$, small $\epsilon > 0$
2.  Initialize $Q(s,a)$, for all $s \in S^+, a \in A(s)$ randomly, and set $Q(s_T,\cdot) = 0$
3.  **for** $k = 0,1,2,\ldots$ **do**
4.      Initialize $S$
5.      **for** each step of the episode $t = 0,1,2,\ldots,T-1$ **do**
6.          Choose $A$ from $S$ using a policy derived from $Q$ (e.g., $\epsilon$-greedy)
7.          Perform action $A$ observe $R, S'$
8.              $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',a) - Q(S,A)]$
9.              $S \leftarrow S'$
10.     **end for** – when $S$ is terminal
11. **end for**
```
Algorithm 5 - Q-learning pseudo-code.

Appendix A.II – Deep Reinforcement Learning

The algorithms presented in the previous chapter are typically referred to as tabular methods, in the sense that they require a matrix to hold a dynamic representation of Q-values. However, the state space grows exponentially with the number of state variables, making tabular tracking infeasible. This is referred to, as the curse of dimensionality. This is where the term deep –

which refers to the existence of multiple processing layers in a neural network – steps in. Instead of storing the values, a Q-value function approximator, a neural network is used.

The use of neural networks instead of simpler function approximations, for instance, linear approximators, is largely justified by that fact that when applied to Q-learning, the approximator greatly suffers from instability and divergence. Considering θ as the neural network's parameters, the action function parametrized by θ becomes $q_{\pi_\theta}(s,a)$.

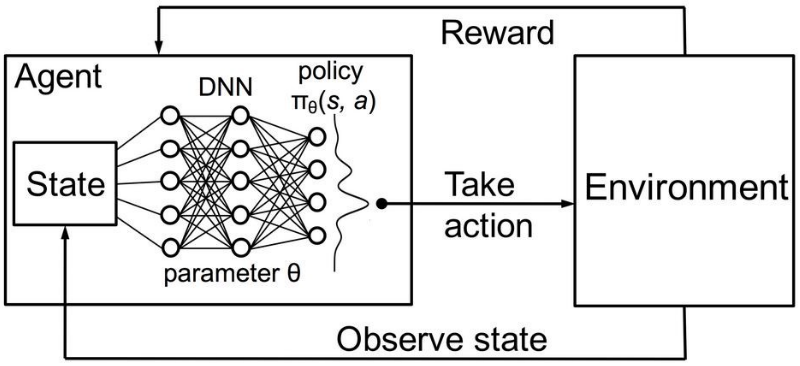

Figure 20 – Schematic depiction of deep reinforcement learning.

Deep Q-learning (Mnih, et al., 2015) constituted a breakthrough in deep reinforcement learning (DRL). The algorithm tackled and managed to overcome some of its tabular predecessor's limitations, by using two innovative mechanisms. The first, experience replay, consists of storing the episode's trajectory on each time step t, $e_t = (S_t, A_t, R_t, S_{t+1})$ into memory $D_t = \{e_0, \cdots e_t\}$. During action function update, a minibatch of independent samples is selected from D_t, and used to train the neural network via stochastic gradient descent (SGD). The mechanism's goal is to reduce the sample's strong temporal correlation, by drawing randomly selected experiences, which yields accurate gradient estimation. The second modification consists in the introduction of a target network, q_{*_θ}, parametrized by θ^-. The independent network is updated less often, specifically, at every C time steps, the target network is updated setting $\theta^- = \theta$, and held frozen for the remaining intermediate period. The use of a second, less oscillating target network, introduces stability in the training process, by attenuating the effect of short-term fluctuations.

By combining the two previous introductions, at every time step t, DQN draws N independent samples from the replay memory D_t, using them to update Q-network's θ, computing the target $Y_t = R_t + \gamma Q_{*_\theta}(S_{t+1}, a)$, and updating θ via gradient of the loss function $L(\theta)$,

$$L(\theta) = \mathbb{E}_{e_t \sim U(D)}\left[R_{t+1} + \gamma Q(S_{t+1}, a, \theta^-) - Q_\theta(S_{t,}A_t)\right]^2 , \qquad \text{(A.10)}$$

where $U(D)$ represents a uniform distribution over the replay memory D.

```
Algorithm Deep Q-Learning
1.  Input: Replay memory capacity $N$
2.  Initialize $D$ to capacity $N$; Initialize $C$, the target update frequency; Initialize both action value
    and target action value functions, $q$ and $\hat{q}$ with random weights $\theta, \theta^- = \theta$, respectively
3.  for $k = 0,1,2, \dots$ do
4.      Initialize $e_1 = \{S, A\}$
5.      for each step of the episode $t = 0,1,2, \dots, T-1$ do
6.          Choose $A$ from $S$ using a policy derived from $Q$ (e.g., $\epsilon$-greedy)
7.          Perform action $A$ observe $R$, $S'$
8.          Set $e_{t+1} = S, A, S'$
9.          Store transition $e_{t+1}$in $D$
10.         Sample random minibatch of transitions from $D$
11.         Set $y_i = \begin{cases} R_t & \text{if episode terminates at } t+1 \\ R_t + \gamma \max_{a'} \widehat{Q_{\theta^-}}(S_{t+1}, a') & \text{otherwise} \end{cases}$
12.         Perform a gradient descent step on $\left[Y_i - Q_\theta(S, A)\right]^2$
13.         Every $C$ steps set $\theta^- = \theta$
14.     end for – when $S$ is terminal
15. end for
```

Algorithm 6 - Deep Q-Learning pseudo-code[12].

Appendix A.III – Policy Gradient

Whereas members of the Q-Learning taxonomy indirectly optimize the policy via estimating the optimal value function, $q_*(s, a)$, Policy Gradient (PG) aims at learning a parametrized policy, without requiring a value function to determine action selection. Policy Gradient represents a policy explicitly, $\pi_\theta(a|s)$, optimizing the policy's parameters θ either directly, via gradient ascent on the reward function $J(\theta)$, or indirectly, by maximizing local approximations of the reward function. The expression $\pi_\theta(a|s)$ can be interpreted as the probability that action a is selected at time step t, given that the environment is at state s, at time step t, parametrized by θ.

$$\pi(a|s, \theta) = \mathbb{P}\left[A_t = a | S_t = s, \theta_t = \theta\right] , \qquad \text{(A.11)}$$

The previous expression can also be represented as $\pi_\theta(s, a)$.

[12] The algorithm differs from the literature, as the original framework was designed for training under computer games, using Convolutional Neural Networks (CNN) to pre-process the images. However, these topics are beyond the scope of the paper, hence, the algorithm's pseudo-code has been adapted.

PG offers significant benefits over DQN. In terms of convergence, value-based methods tend to have large oscillations due to small variations in the estimated action values, can dramatically modify action selection probabilities, PG on the other hand, simply uses gradient ascent to follow the optimal parameters, resulting in smooth policy updates. The newly introduced class of methods is also more effective in high-dimensional spaces regarding the value-based method, being able to cope with continuous action spaces. Whereas value-based methods, at each time step, require computing the value associated with each action, a computationally expensive framework, policy-based methods on the other hand, compute the optimal action directly on θ. Lastly, Policy Gradient methods can learn stochastic policies. The latter leads to two positive side-effects, the obliteration of the perceptual aliasing problem[13] and remove the need to implement a greedy policy method for action selection, such as ϵ-greedy, to handle the exploration-exploitation tradeoff.

Under the Policy Gradient framework, the reward function is given by

$$J(\theta) = V_{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[V_1] \, , \tag{A.12}$$

considering episodic environments, and

$$J(\theta) = \sum_{s \in S} d_{\pi_\theta}(s) V_{\pi_\theta}(s) \, , \tag{A.13}$$

for continuous environments, where d_{π_θ} represents a stationary distribution for a Markov Chain for π_θ, and $V_{\pi_\theta}(s)$ corresponds to the estimated value function of π_θ for state s.

The computation of $\nabla J(\theta)$, required to optimize the parameters θ, is not trivial, as it depends on the effect of the policy on action selection, directly determined by π_θ, and the stationary distribution of states, indirectly determined by π_θ thus, unknown. Policy Gradient Theorem yields the expression which allows computing the gradient of $J(\theta)$, $\nabla_\theta J(\theta)$, which does not involve taking the derivative of the objective function's state distribution d_{π_θ}

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta \, (s, a) Q_{\pi_\theta}(s, a) \, \right] , \tag{A.14}$$

where ln represents the natural logarithm, and the instantaneous reward R_{t+1} is replaced by the long-term action-value $Q_{\pi_\theta}(s, a)$.

[13] Perceptual Aliasing refers to the case of similar or equal states, which require different responses

REINFORCE, or Monte Carlo Policy Gradient, borrows from Monte Carlo – hence its name – in determining how samples are obtained on each time step. The complete return G_t is used as an unbiased sample of $Q_{\pi_\theta}(s, a)$ to determine the policy's gradient to update θ.

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[G_t \frac{\nabla_\theta \pi_\theta(A|S)}{\pi_\theta(A|S)}\right] \\ &= \mathbb{E}[\ln \pi_\theta(A|S)] \text{ since } \nabla \ln x = \frac{\nabla x}{x}, \end{aligned} \tag{A.15}$$

Algorithm REINFORCE: Monte-Carlo Policy Gradient

```
1.   Input: A differentiable policy parametrization $\theta$; step size $\alpha > 0$
2.   Initialize $\theta$
3.   for $k = 0,1,2,\ldots$ do
4.       Generate a trajectory following policy $\pi(\cdot \mid \cdot, \theta)$, $(S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_t)$
5.       for each step of the episode $t = 0,1,2,\ldots,T-1$ do
6.               $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
7.               $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla \, ln \, \pi_\theta(A_t|S_t)$
10.      end for – when $S$ is terminal
11.  end for
```

Algorithm 7 - REINFORCE pseudo-code.

The previous method, although promising, suffers from high-variance. A typical solution to such problem is to subtract a baseline $b(s)$ from G_t, decreasing the variance of the gradient's estimation whilst keeping the update's expected value unchanged. Such variation is called REINFORCE with Baseline, and would modify the gradient ascent, step 7, by subtracting a given value, such as the Advantage function, to be introduced ahead, from G_t.

Vanilla policy gradients tend to suffer from noise and high variance, leading to unstable learning processes, which may result in policy distributions skewed towards non-optimal trajectories. The actor-critic framework tackles said hindrance, by introducing a concept of dual entities, where the critic's responsibility is to estimate the value function's parameters w, $Q_w(s, a)$ or $V_w(s)$, and the actor's role is to update the policy's, $\pi_\theta(a|s)$, parameters θ, in the direction suggested by the critic.

Before proceeding, a new operator, the advantage function, is introduced. The construct attempts to capture to what extent an action better or worse for a given state, in comparison with the alternative actions. The goodness of an action is given by its Q-value $Q(s, a)$, whilst the average value of the actions for a given state, are given by the value function $V(s)$, hence the advantage function is given by

$$A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)\,, \tag{A.16}$$

Advantage functions reduce variability and solve the perceptual aliasing problem, where two similar or equal states require different actions.

The last algorithm presented in this section, is named Trust Region Policy Optimization, or simply, TRPO. Introduced by Schulman, et al. (2017), it is based on the idea that training stability can be achieved by ensuring that the policy parameter updates are limited in size, by introducing a Kullback–Leibler (KL) divergence constraint on the size of the policy update.

The objective function under TRPO is given by

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{old}},a\sim\pi_{old}}\left[\frac{\pi_\theta(s)}{\pi_{\theta_{old}}(s)}\hat{A}_{\pi_{\theta_{old}}}(s,a)\right], \tag{A.17}$$

where θ_{old} represents the policy's parameters prior to the update, $\hat{A}_{\pi_{\theta_{old}}}(s,a)$ represents the estimated advantage function, and lastly, $\rho^{\pi_{old}}$ represents the state's visitation frequency.

The algorithm then seeks maximizing the objective function (), subject to the trust region constraint, which establishes that the distance, as measured by KL-divergence, between the old and new policies, has to be bounded by a given parameter δ

$$\mathbb{E}_{s\sim\rho^{\pi_{old}}}\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot \mid s)||\pi_\theta(\cdot \mid s)\right)\right] \leq \delta\,, \tag{A.18}$$

where $D_{KL}(\cdot\,||\,\cdot)$ represents the Kullback–Leibler divergence, and δ corresponds to the bounding parameter or KL-divergence limit.

Algorithm Trust Region Policy Optimization

```
1.  Input: Initial policy parametrization $\theta_0$; initial value function parameters $\phi_0$
2.  Set the hyper parameters: KL-divergence limit $\delta$,backtracking coefficient $\alpha$, maximum number of
    backtracking steps $K$
3.  for $k = 0,1,2, \ldots$ do
4.      Collect a set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
5.      Compute rewards-to-go $\hat{R}_t$
6.      Compute the advantage estimates, $\hat{A}_t$ based on the current value function $V_{\phi_k}$
7.      Estimate the policy gradient as
            $\hat{g}_k = \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^{T} \nabla_\theta log\pi_\theta(s_t)|_{\theta_k} \hat{A}_t$
8.      Use the conjugate gradient algorithm to compute
            $\hat{x}_k = \hat{H}_k^{-1} \hat{g}_k$
        where $\hat{H}_k$ is the Hessian of the sample average KL-divergence
9.      Update the policy by backtracking line search with
            $\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k$
        where $j \in \{0,1,2, \ldots K\}$ is the smallest value which improves the sample loss and satisfies the
        sample KL-divergence constraint
10.     Fit value function by regression on mean-squared error
            $\phi_{k+1} = \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$
        typically via some gradient descent algorithm
11. end for
```

Algorithm 8 – Trust Region Policy Optimization pseudo-code.

Appendix B – The reinforcement learning policy’s plot as a function of the time remaining until the backtesting process resumes

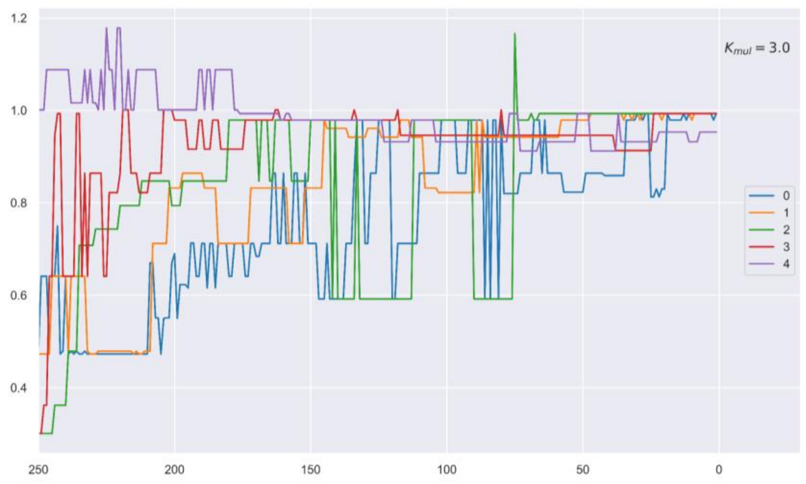

Figure 21 - The proximal policy optimization agent’s policy plot considering the multiplier 3, considering exceedances between 0 and 4.

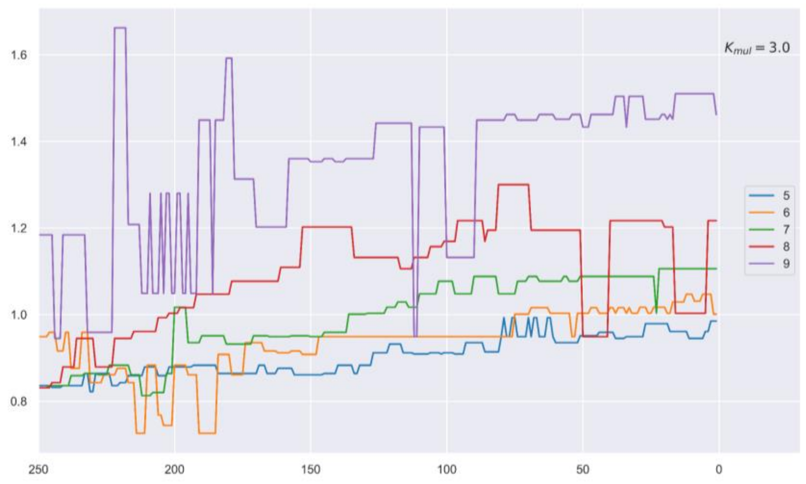

Figure 22 - The proximal policy optimization agent’s policy plot considering the multiplier 3, considering exceedances between 5 and 9.

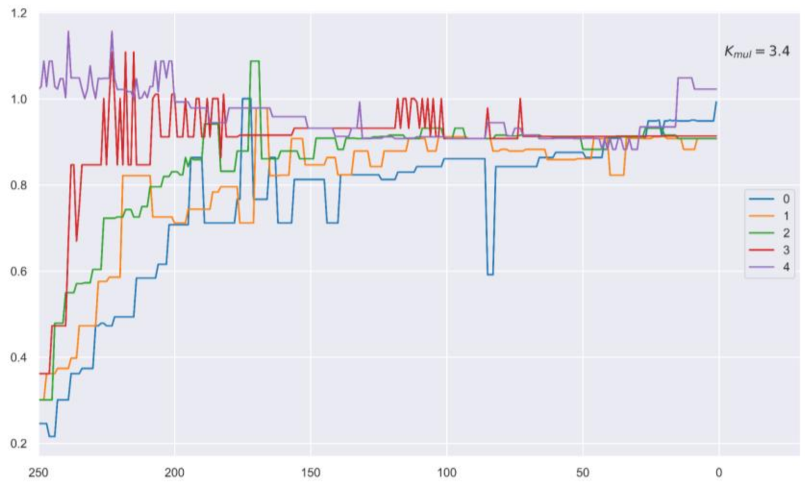

Figure 23 - The proximal policy optimization agent’s policy plot considering the multiplier 3.4, considering exceedances between 0 and 4.

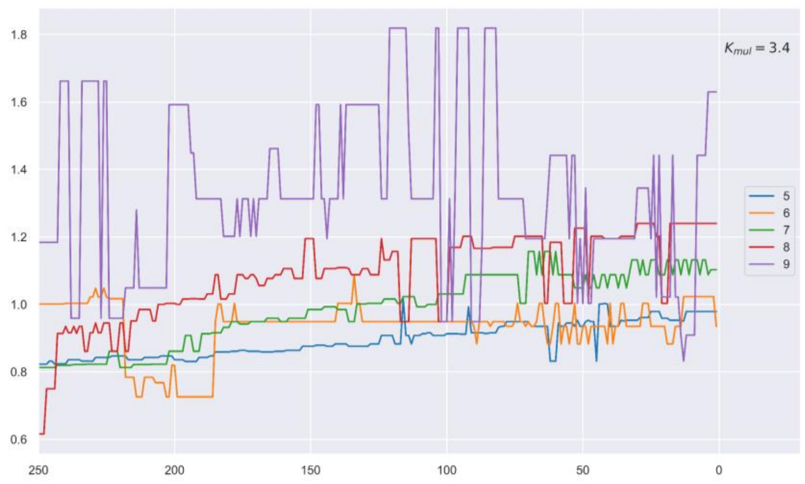

Figure 24 - The proximal policy optimization agent’s policy plot considering the multiplier 3.4, considering exceedances between 5 and 9.

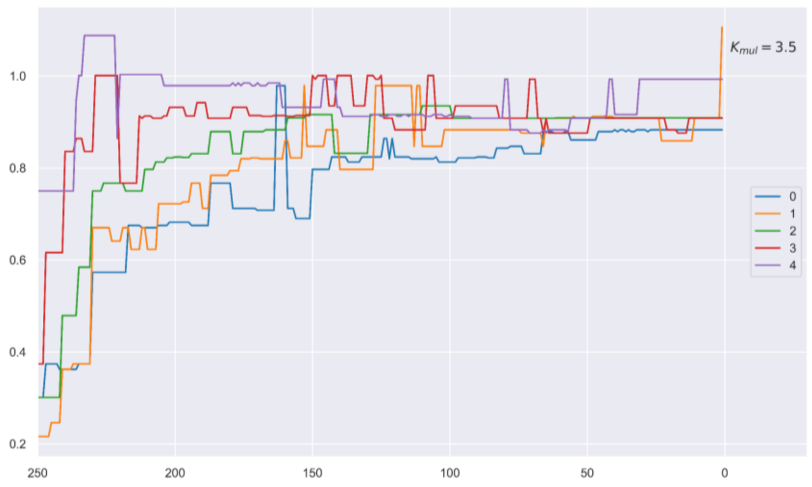

Figure 25 - The proximal policy optimization agent’s policy plot considering the multiplier 3.5, considering exceedances between 0 and 4.

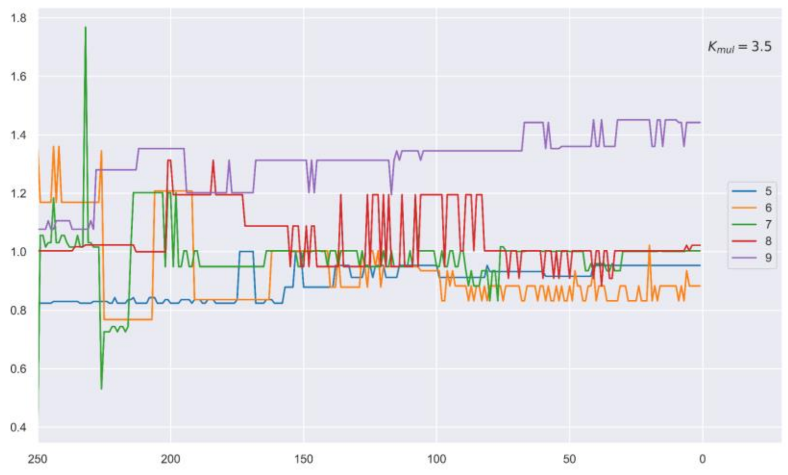

Figure 26 - The proximal policy optimization agent's policy plot considering the multiplier 3.5, considering exceedances between 5 and 9.

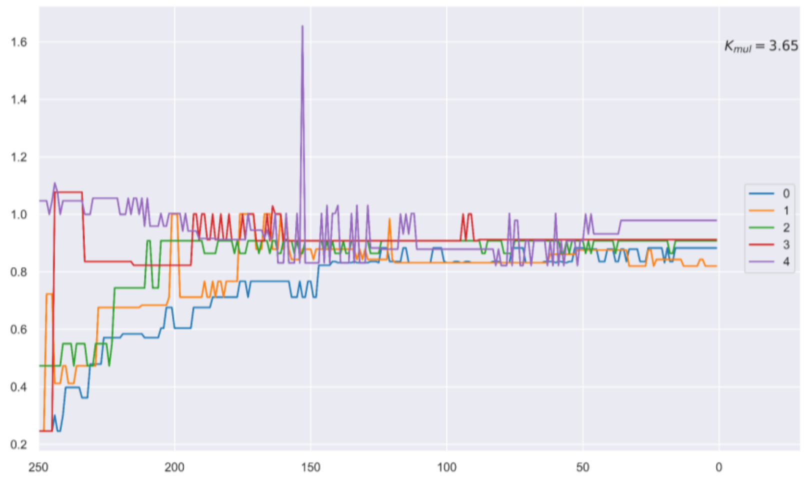

Figure 27 - The proximal policy optimization agent's policy plot considering the multiplier 3.65, considering exceedances between 0 and 4.

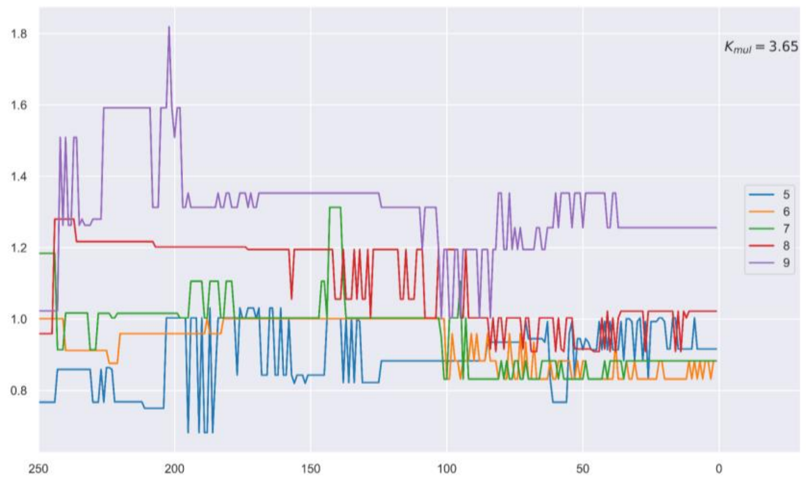

Figure 28 - The proximal policy optimization agent's policy plot considering the multiplier 3.65, considering exceedances between 5 and 9.

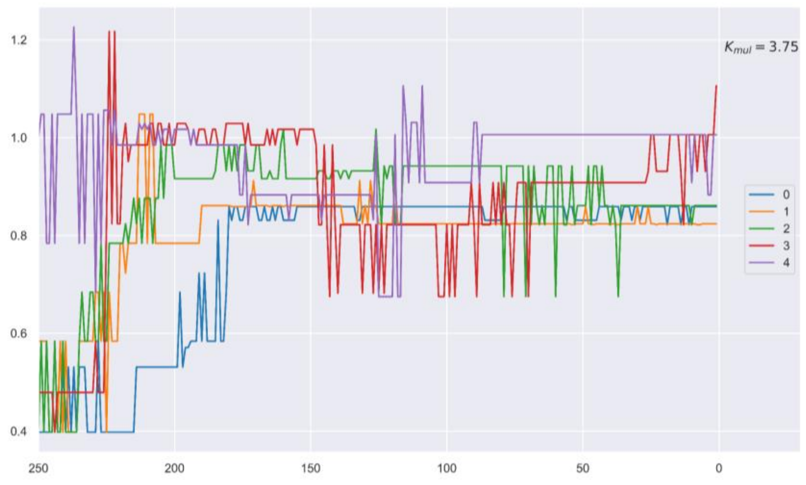

Figure 29 - The proximal policy optimization agent's policy plot considering the multiplier 3.75, considering exceedances between 0 and 4.

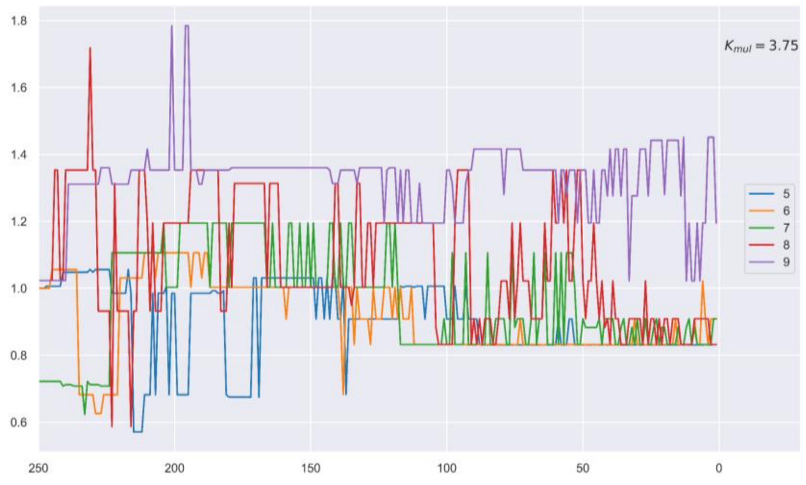

Figure 30 - The proximal policy optimization agent's policy plot considering the multiplier 3.75, considering exceedances between 5 and 9.

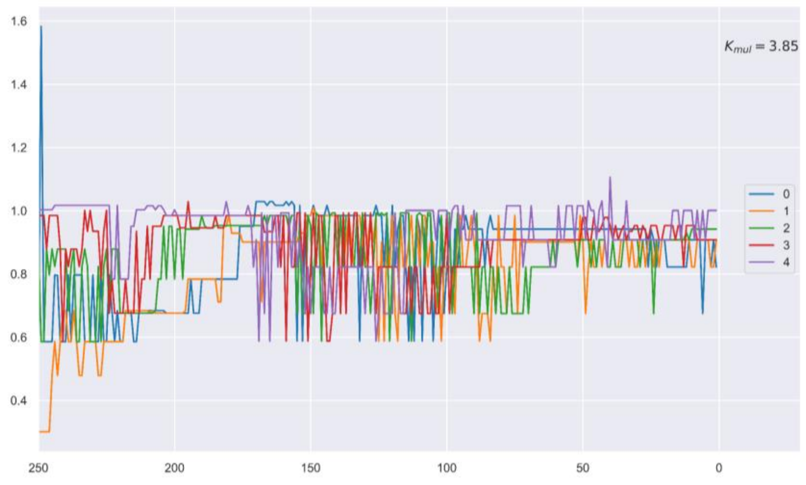

Figure 31 - The proximal policy optimization agent's policy plot considering the multiplier 3.85, considering exceedances between 0 and 4.

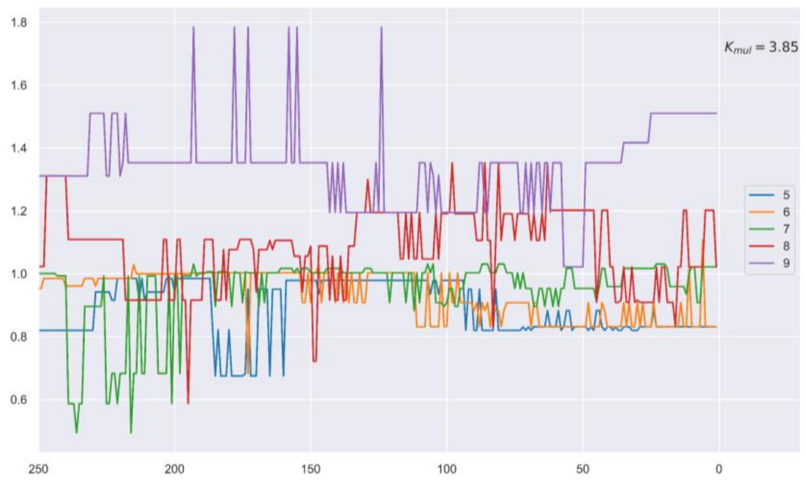

Figure 32 - The proximal policy optimization agent's policy plot considering the multiplier 3.85, considering exceedances between 5 and 9.

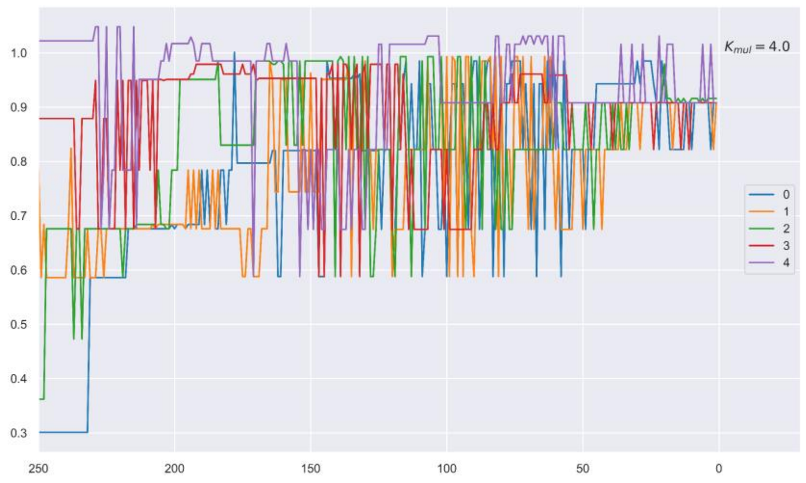

Figure 33 - The proximal policy optimization agent's policy plot considering the multiplier 4, considering exceedances between 0 and 4.

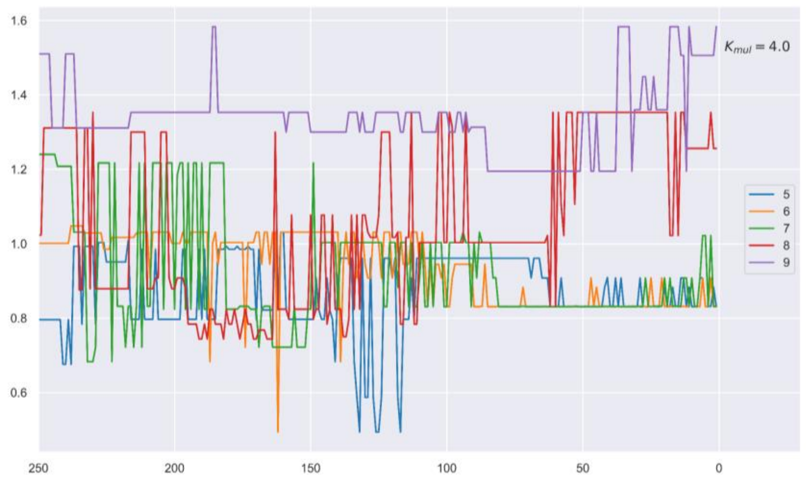

Figure 34 - The proximal policy optimization agent's policy plot considering the multiplier 4, considering exceedances between 5 and 9.

Appendix C - Source Code

The author has chosen to publish the source code which led to the findings present on this work under his public GitHub repository at github.com/guilherme-b. The repository includes both the reinforcement learning’s environment and the Monte Carlo simulation framework.