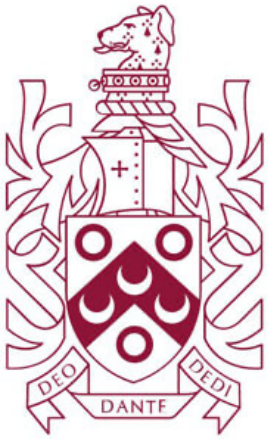
EXTENDED PROJECT

QUALIFICATION (EPQ)

SUMMER 2025



Pupil Name: Augustus David Roberts

EPQ Title: The Solutions and Significance of the Multi-Armed Bandit Problem

Project Type: P303 – Artefact

Centre number: 64390

Candidate number: 4565

Contents Page

Table of Contents

Abstract

The Multi-Armed Bandit problem was first introduced to me in a computer science lesson in the first few weeks of the academic school year, as a problem aimed to promote engagement in machine learning. At that time, I was competent in programming, but I had little knowledge on machine learning, but I found the problem extremely interesting, so my curiosity prompted me to explore further. Prior to starting my EPQ I began reading into machine learning as my fascination developed. I found myself spending almost every minute of my free time listening to podcasts, reading articles and books on machine learning and artificial intelligence. Three weeks on, I had read the books 'CODE' and 'Life 3.0', alongside sections of many other books, watched the documentaries 'Coded Bias', 'The Social Dilemma' and 'AlphaGo' and a plethora of Ted Talks and other YouTube videos. All this research continued to fuel my curiosity more and develop my understanding of how crucial education of technology is, prompting me to use the opportunity of the EPQ as a tool to express this.

This project examines the significance of the Multi-Armed Bandit problem, and my artefact provides programmed solutions to the hypothetical problem itself, with outputs showing results of these algorithms. The artefact also provided other programmed content in aid of proving the significance of the problem. The main body of the dissertation is divided into three sections: Solutions, Theory and Application. The aim of these sections was to provide robust evidence

to support the title of ‘The Solutions and Significance of the Multi-Armed Bandit Problem’, using a combination of third-party sources and my programmed artefact.

My main achievements in this project regarding solutions to the problem were: the Upper Confidence Bound approach and the Hybrid approach. My chief finding regarding significance was that these solutioned proved to be directly applicable to a variety of fields, such as high-frequency trading shown by my creation of a model trading algorithm.

Recommendations for the reader

As I programmed my artefact and the problem requires the use of technical language and jargon, I have created a Glossary which I recommend the reader visits when reading my dissertation and discovering said language.

Throughout the main body, I frequently reference the Appendix as my artefact is contained within different screenshots of the outputs of my code, and I have included many photos and different programs, vital to understanding my content. Therefore, I recommend making the Appendix readily available and visiting the referenced Appendices every time I mention them.

Introduction

Problem Description

The Multi-Armed Bandit (MAB) problem is named after a hypothetical thought experiment where a gambler is faced with multiple slot machines, each returning unknown rewards. Each machine corresponds to an action with rewards drawn from a unique probability distribution, where the mean reward (bias) of each machine, at the centre of the distribution, is unknown. The aim of the gambler (Bandit) is to maximise his rewards after taking a set number of actions, by recognising the machine with the greatest bias and selecting it the greatest number of times.

The solution to solve the MAB problem is balancing two key terms, exploration and exploitation:

- Exploration: Trying different machines to gather more data to estimate their mean reward (bias).
- Exploitation: Choosing the machine that is currently believed to offer the highest return, based on an estimation of its mean reward.

The challenge of estimating the bias of each machine arises as their rewards are distributed using normal (Gaussian) distribution centred around their bias, meaning that the reward returned by the machine often does not reflect the true means accurately.

Aims Overview

In my artefact I have developed several approaches to this problem, each varying in practicality, complexity and performance under different conditions. My artefact itself is a combination of programs, providing solutions to the problem and outputting graphs to representing their results, as well as algorithms designed to prove the significance of the problem such as an automatic trading algorithm. My artefact aims to provide solid proof for

each one of my MAB solutions, which I derived from both ranges of sources from experts and innovations of my own. I aim to explain the mathematics and complexities behind this hypothetical problem, and to simplify and relate this problem to real world examples such as an individual choosing which restaurant to visit. I aim to emphasise the significance of this problem using examples where similar algorithms are used for high frequency trading, social media and advertisement online and explaining the impact of these algorithms in these fields.

Structure Overview

Whilst I have briefly covered the contents of my main body in the previous paragraph and on the contents page, due to the slightly unorthodox style of my EPQ, being an artefact constructed via programming, I have been limited to supplying only screenshots of my results rather than direct access to run the programs. The inputs, the information the user is promoted to enter by my programs, and the output, the information derived from those input, have been included in my Appendix, and referenced appropriately. My main body itself, alongside the introduction, has been designed to accommodate for reader who are completely new to the topic, and structured to ease the reader into an interpretation of the problem. Moreover, I have included a few photographs in the Appendix visually representing the problem, which I found helpful when first coming to terms with it. To surmise, I am going to explain each approach I have taken step by step under the ‘Approaches’ subtitle, followed by an explanation about the relevance and depth of mathematics that can be involved with the problem. Then I aim to explain the connection of the Multi-Armed Bandit Problem to real world scenarios and uses in the real world, then subsequently conclude my discussion with my most effective solutions, and my best arguments for the problems significance. I will emphasise my primary successes, failures, limitations and experience gained from this large project in my evaluation.

Literature Review

Literature Overview

Given the limitless scope of my artefact on 'the multi-armed bandit problem', I have been led to research a plethora of sources of a various nature. Additionally, due to the relatively recent advent of this enduring mystery of a problem, I have been able to access a vast range of primary sources. Moreover, these solutions are derived from universal mathematical theories dating back hundreds of years, which means that I can access a range of secondary sources too. I have focused my research into sources such as reports, academic articles and lecture videos, in order to cover a range of information from statistics, opinions and evaluations, located in Google Scholar, YouTube and standard browsers. Since my artefact is built using code, I have also found a range of helpful resources to develop my programming abilities in order to tackle this problem in the physical sense. This review aims to evaluate the provenance and adequacy of these resources as an aid to my EPQ artefact, emphasising their key takeaways, limitations and comparisons to other sources. I have structured the review by initially explaining the sources I have used to develop my ideas and understandings, under the sources subtitle. This is followed by the materials subtitle, where I will refer to the sources I have used as materials for building my artefact itself.

Sources

One type of source I have discovered entails primary resources from authors who attempted to solve the Multi-Armed Bandit problem themselves. Such resources were particularly crucial as they initiated creative thinking by offering a point of departure of many solutions to the problem. An example is Weng (2020), an article discussing exploration strategies in a

way that breaks down complex ideas into straightforward and easy to picture concepts. This material was especially helpful due to its simplicity compared to others that previously overwhelmed me with complex mathematical proofs well beyond my starting point of comprehension. Weng's description of the SoftMax technique, with the use of Boltzmann's distribution and variable 'temperature' instead of reducing epsilon, to generate weighted distribution, sensitised me to new avenues of inquiry that directly influenced my project. The authority and reliability of this source is also assured by the qualification of the author. Lilian Weng is a PhD student at Indiana University Bloomington School of Informatics and Computing, which gives her strong academic credentials in reinforcement learning. Secondly, the content of her research aligns with other reliable methods I encountered in research. In general, this resource not only contributed to my theory but also made me more precise in terms of the practicality of my project.

Another helpful primary source was (Silver, 2015), a Computer Science undergraduate degree course PowerPoint presentation for the Reinforcement Learning module. This source provided a fantastic insight into mathematics behind various solutions to the problem, allowing me to understand and compare algorithm performance. For instance, the slides had used the term 'regret', which I have used as a metric for comparing the performance of different algorithms. The brevity and visuality made easy incorporation of key concepts and application in real life scenarios. The disadvantage of the source was that it was not detailed. While it may suffice to outline the fundamentals, it failed to consider the wider implications of the matter, which I had aimed to include in my project. Other than that constraint, the source is faultless. David Silver is a research scientist at Google DeepMind, which is one of the leading artificial intelligence research facilities globally. In addition, the module is included in the University College London UCL Computer Science course syllabus, an

indication of academic credibility. The paper was very crucial in the attainment of a good theory in my project.

Another type of source utilised operated on secondary sources from a combination of existing work to define a specific formula or concept under which I have operated with in my artefact. Still in regard to consideration of the secondary literature, (Chen, 2024) also provides a crucial definition of the Gaussian (Normal) distribution, a universal statistic and component in some of the algorithms to solve the multi-armed bandit problem. This was a very useful source because it gave a simple explanation of the formula and uses, which informed me on how to use this distribution to estimate reward probabilities and uncertainties in my program. The article by Chen also fills in the gap between theoretical statistics and real-world application, providing examples that were most directly applicable to the bandit problem. With regards to reliability, the source is reliable because it is a direct copy from Investopedia, one of the top resources for studying statistics and finances. Furthermore, James Chen has written about finances for over 20 years and is an editor as well with advanced concepts which are explained simply for ease of understanding. However, one of the downfalls of this source is that it is a generalist source, does not go into the intricacies of using Gaussian distribution in reinforcement learning. Nevertheless, its readability and introductory level information that it contains made it a treasure trove for information when it came to understanding a fundamental aspect within my artefact. Generally, this source is useful due to its usability and usability in using it in an effort to assist me in applying the statistical basis of my program.

The Vermorel and Mohri (2005) paper was a goldmine, providing an exhaustive comparison of multi-armed bandit algorithms like epsilon greedy, UCB, and Thompson Sampling. Such a reference proved extremely useful in guiding my direction, especially their empirical

comparison of algorithm performance by means of cumulative regret. The paper also provided pleasant visualisations, which helped me compare results correctly and further hone my implementation. One of the source's strengths is that it gives clear mathematical proofs, which enriched my knowledge and justified the strategies I applied in my project. Its age, published in 2005, limits its use to contemporary developments in reinforcement learning. Some of the proofs were also extremely detailed, and extra research was necessary in order to understand the applicability to its full extent. Although it is a secondary source, the credibility of its sources is guaranteed by the fact that it was published in the proceedings of the 'European Conference on Machine Learning', which is a reputable academic journal, and the experience of its authors, Joannès Vermorel and Mehryar Mohri. It has some weaknesses, but this paper had a good theoretical and comparative framework that assisted in developing my artefact significantly.

Li, Chu, Langford, and Schapires (2010) contextual bandits problem research paper provides a detailed account of how the basic multi-armed bandit problem can be extended to problems where the decision is made in the presence of contextual information. The source was useful because it made me realise the concept of contextual features while being crucial in realising how bandit algorithms can be utilised in real-life applications, like personalised suggestions or adaptive pricing. The explicit description of algorithms like 'Lin UCB' and contextual epsilon-greedy and their theoretical guarantees made it easier for me to attempt to implement contextual variables into my own artefact. I believe, perhaps the strongest aspect of this paper is its dual focus on theoretical rigor and empirical testing. The authors give both diligent proofs to establish their algorithms and empirical data showing their performance in a variety of contexts. This balance was particularly helpful in helping me test and iterate on my implementation. One of the paper's limitations is that its analysis of computational

complexity is dependent on the availability of vast computational resources. This hindered the direct implementation of some of the techniques as it was limited by my project environment. The credibility of this source is noted through its authors, as all the authors are leading researchers in machine learning and John Langford is one of the leading reinforcement learning experts. Additionally, the journal appeared in the Journal of Machine Learning Research which is a reputable academic firm. In summary, this source presented a great amount of data on an extension of the bandit problem, enabling me to make my project broader and applicable.

Another vital source that I came across during the course of my studies was Auer, Cesa-Bianchi, and Fischer (2002) article. In the article, they proved the theoretical basis of the UCB (Upper Confidence Bound) algorithms. The article explained how to address the problem of exploration-exploitation trade off. This was especially vital for adversarial environments. They concluded UCB1 and its variants in the article, which helped me refine my understanding of adaptive control of exploration in my work. It was different from Weng (2020), where he emphasised the intuitive and applied aspect more. Instead, Auer et al. favoured a theoretical and formal, proof-focused path. It helped me convince myself that my program was good. The only disadvantage of the source was its complexity. The source also assumed a great amount of mathematical background, so it was difficult to apply directly. However, the reliability of the authors proved evident as they worked as professors in top universities, which led me to trust the theoretical basis they established. This source influenced my project, in which I adapted the UCB1 formula to create a more effective hybrid approach by adding the exponential decay.

Also, Sutton and Barto (2018) in Reinforcement Learning: An Introduction were an important source in the case of learning more general reinforcement learning schemes, including bandit problems. Unlike sources which had merely discussed the multi-armed bandit problem, Sutton and Barto placed the problem in the context of a more general reinforcement learning scheme, illustrating the relationship to Markov Decision Processes and Q-learning. This was instrumental to an understanding of my research in the broader AI context. They also mentioned regret minimisation, which was another reason for applying regret to my project as it was also discussed by Silver (2015). The description of the algorithms was simpler compared to Vermorel and Mohri (2005), where additional material needed to be referred to in order to understand the proofs. Sutton and Barto's text was much longer, however, so the chapter on multi-armed bandits was relatively brief. It provided the necessary foundation, but it did not have as much detail as Vermorel and Mohri's empirical comparisons.

Finally, a significant contribution to my research came from Lattimore and Szepesvári's (2020) "Bandit Algorithms," which systematically explores various multi-armed bandit strategies with a strong emphasis on modern advancements such as Thompson Sampling and Bayesian bandits. This book filled an important gap by bridging older theoretical works, like Auer et al. (2002), with contemporary developments in the field. One of its strengths was its wide range of coverage of Bayesian approaches, which was contrasted with the frequentist approaches utilised in the majority of my other sources. The comparison between Bayesian inference and UCB approaches was especially helpful, further reinforcing my grasp of why different approaches are suited for different environments. Unlike Chen (2024), which provided a simple statistical overview of Gaussian distributions, Lattimore and Szepesvári had an inherently deeper and more formal mathematical treatment of probabilistic approaches in bandits. The only setback was that the book, being rather theoretical, at times was difficult

to directly implement in a real coding environment. However, its findings helped shape the theoretical basis for my selected methodologies and offered pathways to future refinements in my artefact.

Materials

One of the most important parts of my project development was using a range of programming resources to advance my coding skills and debug advanced problems. Of the most effective sources were Stack Overflow, W3Schools, GeeksforGeeks, and Real Python. All of these websites served a different purpose in my learning experience, with varying strengths and weaknesses that guided my solution to the Multi-Armed Bandit problem in my artefact. This section critically evaluates these sources for utility, credibility, the degree of explanation, and how well they were suited to my work.

Stack Overflow (Stack Exchange, 2024) was a great debugger and problem-solver source in real-time. As an interactive question-and-answer website, it provided user-based solutions whereby experienced web developers provided their opinion on some coding problems. One of the best assets of Stack Overflow was that it provided multiple solutions to a single problem, usually with various reasons from different viewpoints. This allowed me to weigh out multiple approaches and select the optimal one for my project. Second, the upvote/downvote system helped me to gauge the validity of answers. But one of its drawbacks was inconsistency in quality, as some of its responses were well-referenced and well-explained, while others were hollow or included outdated practices. In comparison to

W3Schools, which includes tutored lessons, Stack Overflow is more solution-oriented and is best utilised to debug real errors rather than learn background information on programming concepts.

W3Schools (Refsnes Data, 2024) proved to be an excellent source for learning syntax and programming concepts, especially for Python. Its unique strength is in its easy-to-follow structure as a beginner resource, with live code examples, which enabled me to test out functions prior to applying them within my project. In contrast to Stack Overflow's fragmented solutions, W3Schools is structured, and hence progressive learning topics are easier. Although W3Schools is better in terms of accessibility and learnability, it is sometimes criticised for oversimplification. A couple of more complex topics, reinforcement learning and multi-armed bandit algorithms, were not adequately explained to my liking in enough detail. This is compared to GeeksforGeeks, which is more technical and detailed in its examination of algorithms.

GeeksforGeeks (GeeksforGeeks, 2024) was highly informative in the understanding of implementation of algorithms such as reinforcement learning basics and optimization algorithms that are useful for my project. GeeksforGeeks provided explanations fully, at times breaking down hard algorithms stepwise. The pseudocode and the other programming illustrations also enabled me to solidify my own implementation of code for Upper Confidence Bound and Thompson Sampling algorithms in my artefact. GeeksforGeeks offered more theoretical knowledge than W3Schools and therefore was more suited for intermediate to advanced programmers. Its only flip side was that certain articles lacked clarity in real-world usage and therefore required me to refer to Stack Overflow or other sources to debug.

Finally, Real Python (Real Python, 2024) had to be consulted to further polish my Python coding style and good practices. Unlike the other three resources, which are community-based or tutorial-based, Real Python provides expert-written articles and video courses for a systematic in-depth learning of programming principles. This was particularly helpful in optimising my code to be more efficient and readable, which had a direct impact on optimising my artefact. The best part about Real Python was its hands-on, project-oriented methodology that meets my EPQ goal to submit a functioning program rather than dry research. Real Python was fine for making my programming workflow better, but the downside was the accessibility—the superior content was on a pay wall, so it was not fully as easy to access as GeeksforGeeks or W3Schools.

To compare sources, W3Schools and GeeksforGeeks both provide tutelage through organized tutorials, but W3Schools is better for beginners, while GeeksforGeeks goes in-depth with algorithms. Stack Overflow and Real Python both provide solution to problems but vary in nature—Stack Overflow is collaborative with varied answer quality, while Real Python provides courses guided by experts. They together created a balanced environment that enabled me to learn, debug, and optimize my programming approach successfully. Being able to verify information coming from varied sources made my artefact stronger, with provision of precision, effectiveness, and code solidity in development.

Design Choices

Since the Multi-Armed Bandit Problem is a slightly unorthodox artefact to create for the EPQ qualifications, and often I program without a second thought as to why I make the design

choices that I do, I have been influenced to take a new perspective on the significance on my design choices in programming.

Initially, I had to choose an Integrated Development Environment (IDE) which is a software platform which provides the ability to execute and edit my code. Most common features of a standard IDE include a code editor, a debugger, a compiler or interpreter which are essential for developing a program. Upon the decision of which IDE to utilise for my project, I factored in three main features: compatibility with libraries, familiarity with the interface and a range of accessible tools for editing my program. Considering this, I decided to use Visual Studio Code (VSC) for my project, due to its support for a large range of libraires and compatibility with Python (Version 3.11.2). Additionally, I am very familiar with its interface as I have done many projects and worked with it before, so I could confirm that it was easy to use. Also, VS contained features and functions like code linting, auto-completion, and debugging, would allow me to build my algorithms more efficiently. The reliability of Visual Studio Code is reflected in its widespread use, with over 14 million monthly users, representing approximately 70% of programmers globally, it is trusted by professionals and students alike.

The next design choice I made involved the planned structure of my programs. Since I predicted I would spend the majority of my time programming building upon already existing algorithms or developing new approaches to solve the problem, I decided that I should build the infrastructure for the main program initially before developing approaches. This meant that I could implement my different algorithms as functions and test them using the main program, using procedural programming as my main programming paradigm. I decided to opt for a procedural programming approach as when I started my EPQ I was relatively unfamiliar with other programming paradigms, such as Object-Oriented Programming, and procedural programming suited the functionality of the program adequately.

My choice of my two libraries, NumPy for mathematical functions and array handling and Matplotlib for plotting my graphs, were recommended by the reliable authors of (Harris et al., 2020) and (Han and Kwak, 2023). They promoted their ease for programming and efficiency in functionality, with NumPy providing me with functions such as ‘np.argmax()’ returning the highest valued item in a data structure and Matplotlib providing the efficient, presentable and clear graphs shown for many of my outputs shown in the Appendix.

Main Body

Solutions

My initial topic of discussion for the main body involves the explanation of each of the solutions I explored in a chronological order. Each solution represents a different approach to solving the Multi-Armed Bandit problem, starting with my first simple solution ‘exploroit’ and building upon each one as I learn and develop new ideas. For each solution I aim to answer questions such as: Why have I explored this approach? What are the failures and successes of this approach? What have I learned from exploring this approach? I will use additional content explaining how each approach works mathematically and programmatically, including my decision-making processes. I will provide visual evidence of my work in the Appendix, including graphs, my source code, my outputs and other relevant information. This section of my main body aims to progressively introduce and represent my journey of discovery and innovation, providing clarity and easing the reader into a shared comprehensive understanding of these solutions.

Exploroit

The first approach to this problem was designed with the principal aim of understanding the problem, rather than coming to an immediate solution. This approach was undoubtedly inefficient; however, I managed to learn and understand the problem much more after mastering the fundamentals. The first simplified approach I took, I have coined as 'exploroit,' an approach where I designate a select number of actions as pure exploration followed by pure exploitation. This involves trialling each machine a certain number of times, then taking the highest mean rewards from the pulls of each machine and exploiting it thereafter. This way, the average is evaluated after a 'practice phase', where the bandit consciously sacrifices optimisation to gain data to estimate the optimal action to exploit indefinitely, which compensates for the previous sacrifice of regret. Appendix 2 provides a visual representation of this approach.

While this approach is somewhat acceptable for solving the problem, it is extremely naïve and unreliable with varying performance under certain input conditions. For instance, if the user were to input the exploration phase to last one spin per machine before exploiting, the probability of not selecting the best action is relatively high due to the limited data. This means actions of regret will continue to occur if the best action is not identified. Moreover, while there is a constant probability of action regret for a set number of practice spins (in this case, 1) and a set number of machines, an increase in the standard deviation (variance) will amplify numerical regret if the best action is not taken. (See Appendix 1) to visualise the empirical regret incurred from the exploration phase, showing a decrease in regret in a logarithmic fashion as the average reward tends to the more optimal means discovered. As shown in Appendix 1, despite the exploration phase of 100 spins per machine, the algorithm still failed to recognise the optimal action (V*), highlighting a large failure of the algorithm. For this algorithm, it can be deduced that the probability of choosing the best action is derived from the

number of practice spins divided by the number of machines. To find the best action for any number of machines, it would seem appropriate to maximise this value of practice spins. However, while increasing the number of practice spins would raise the probability of estimating the best action, there needs to be a cut-off; otherwise, there would be an infinite level of exploration, and numerical regret would increase in a linear fashion to infinity. Likewise, if a greedy approach was taken, allowing only one practice phase and then exploiting thereafter, there remains a significant risk of misidentifying the best machine due to insufficient exploration, with an extreme assumption that the reward distribution is an accurate representation of the means. This emphasises the probability of action regret for any increase in the number of machines or actions. Another weakness of this algorithm is the requirement for the user to estimate and define the cut-off between exploration and exploitation, which is impractical and could be improved if the algorithm were to deduce this balance itself. Moreover, even if the algorithm or individual were to calculate an accurate balance of exploration and exploitation, this algorithm would be extremely impractical for application to real world scenarios. For example, neither the user nor the algorithm is aware of how many times an individual is going to scroll posts on their platform, therefore giving no way to calculate an efficient balance. I have learned a great deal from this approach as it underscores the question of balancing exploration with exploitation, as it visualises the importance of this balance. This approach also aided me in understanding the problem itself and how I can improve areas in future solutions. Additionally, despite the many failures of this approach, in theory, this approach does have a positive performance as it can recognise the optimal machine with consistency or gets close to the optimal machine given a suitable exploration phase on input. Moreover, my failures of this approach have helped me develop a better understanding of the requirements I need to undertake to refine my artefact. Therefore, these finding led me

to explore a more dynamic balance rather than a polarised one, encouraging me to research my next approach, upper confidence bound.

Upper Confidence Bound

The second approach I decided to explore is known as the upper confidence bound approach, (UCB). I was influenced to explore this approach by experts and sources, such as (Silver, 2015) and (Li and Lihong, 2010), as they all promote the efficiency of this algorithm. Unlike my initial 'exploroit' approach, which rigidly separates exploration and exploitation, I decided to explore the UCB algorithm as it dynamically balances these actions by use of an upper confidence interval for each action's estimated reward. This allows accountancy for both the estimated reward and the uncertainty of each action. This upper confidence bound changes in such a way that as the number of times a machine or action is used increases, the bound decreases, thereby reducing its probability of selection. The algorithm selects the next action to take via the application of its formula (See Appendix 3). The formula takes the average reward for that action and apples the formula for the upper confidence bound to it, then multiplies that value by a constant. The machine chosen / action taken, is the action with the highest value after the formula is applied to the information on this machine. This is significant in context of my advancements in my solutions as this method of choosing what action to take allows for fair representation of every action, promoting a dynamic balance between the two states, exploitation and exploration, without too much sacrifice of optimal rewards. Whilst this formula is universal, it is often represented with a differing multiplying constant, often simply 1 or values in the range of $0 < \text{constant} < 2$, shown by (Bandit Algorithms, 2016). However, whilst many credible and efficient sources support this approach such as (Silver, 2015) and (Weng, 2020), I believed that the constant could be improved for a theoretical environment. The constant is simple in the initial UCB formula as it is tailored for application to real world

algorithms, for example Facebook and Google use this formula to tailor advertisements to specific users. I will expand upon how they do this in the application section. Since the Multi-Armed Bandit problem can be solved in a hypothetical environment, ignoring the algorithms utility, my eagerness to explore this problem led me to investigate theoretical approaches to 'ace' this problem directly. Having researched around various constants, I used with the help of academic articles and posts from the sources of (Gamarnik, 2010; Cohn, 2015) and (Baeldung,) regarding logarithmic time complexities. I decided to formulate a constant which is dependent on future information, in this case the number of spins or actions that will occur. Obviously, this removes ideas of application, however in this hypothetical environment, the approach performed far beyond my expectations, providing me with the most efficient and reliable long-term approach, not only stemming of the universal upper confidence bound formula but emphasised by my discovered constant. Numerical proof of this effectiveness can be found in Appendix 4, outlining the different constants and their performances. The primary significance of this constant lies with its ability to adapt the formula for action selection to tailor for the expected number of spins, meaning that if the number of spins is larger the confidence bound will initialise larger in order to promote exploration in order to minimalise action regret, likewise in a shorter timeframe, this exploration bound will initialise as a smaller value to minimalize numerical regret via excessive exploration. Having learned the principles of this approach from (Silver, 2015) and (Li, Lihong) and developing it beyond their suggestions, I can proudly claim this achievement as the most successful in my EPQ journey. For direct proof of this outstanding success of my upper confidence bound approach, (See Appendix 7), showing the average reward over time exceeding all other approaches. I concluded from this approach that a dynamic balance of exploration and exploitation is vital, and their distribution through the program is the most important factor of success. Moreover, this format of applying a formula, in this case the confidence bound, is an efficient way to

handle a dynamic balance whilst accounting for all possible actions. However, the formula for this approach allows the data collected to heavily dictate the balance between exploration and exploitation. To refine my artefact, I next aimed to source a formula which was less reliant on this factor and instead have a more predicable balance, leading me to explore the epsilon decay approaches.

Linear Decay

Following my research from the Upper Bound Confidence approach, considering the new insights I had discovered alongside research of other approaches, I was led to exploring a new approach known as linear decaying epsilon, which slightly varies and generalises the approach to exploration decay. This approach works as an epsilon value (ϵ) is decayed in a linear fashion prior to each action. A random number is generated in the range 0<rand<1. If the random number is greater than the epsilon value, exploitation occurs to the estimated highest meaned action. Otherwise, a random machine is selected and pulled. This differs from the bounded approach as the epsilon value, dictating the probability of exploitation and exploration, is generalised to essentially decrease all exploration in a linear fashion, rather than decreasing bounds for individual values that dictate the balance (Bandit Algorithms, 2016; Lee, 2023; Stack Exchange, 2023), (See Appendix 21).

This approach is an extension of the original ϵ-greedy approach I discovered from GridflowAI (2023), where the ϵ value is not decayed but remains constant. I decided to expand upon this approach as this leaves the aspect of dynamic balance of exploitation and exploration up to probability from the uniform randomly generated comparison value, with a fixed ϵ value found to be optimal at 0.5% (GridflowAI, 2023), (See Appendix 22). By decaying this ϵ value, this promotes a decrease in exploration in a linear fashion, meaning that there is a higher probability

of discovering the optimal or empirically close-to-optimal action initially before increasing greedy exploitation (Weng, 2020; Vermorel and Mohri, 2005). However, upon testing this algorithm, I discovered after repeating the algorithm with the same machines and set number of pulls, there is an extremely high dependence on the initial exploration phase and the cut off for exploration is too rapid, as noted in my script for the code (See Appendix 24). This is shown by the varying performance of the algorithm (See Appendix 23). To comprehensively test this algorithm, I decided to repeat the algorithm 70 times to gain an estimate of its true performance (See Appendix 25), discovering that whilst the algorithm does tend to prove moderate efficiency, with a mean average reward around 1.5, a modal average reward of around 1.6 and a medium average reward of around 1.5. However, four outliers in the range of $0<n<1$ prove inefficiency and unreliability of this method. To check for trends in this data's outliers, I repeated the algorithm once more with 100 repeats (See Appendix 25), again with a similar average and 5 outliers centred around 0.3. From these trials I vaguely estimated an inefficiency rate of 5% derived from the 5 of 100 repeats (5%) and 4 of 70 repeats (5.71%) being outliers. When working with my approaches I was assured of the importance of anomalous results (Esfandiari, (d/m)), as they cannot be disregarded and they are extremely useful to emphasise room for improvement, hence why I began to explore exponential decay as an alternative.

Exponential Decay

Exponential decay, opposed to linear, was marketed to me via a range of sources such as (Chen, 2010), for its improved dynamic balance resulting in an improved solution. The principal idea of exponential decay is to initially reduce the epsilon value slowly, increasing that rate of decay as time passes, allowing a more sufficient exploration phase, whilst gradually introducing exploitation. The difference between the exploration phases in linear and exponential decay can be seen in Appendix 27, with the linear jumping to a faster conclusion of the optimal action and the exponential showing the logarithmic looking transition between the two states.

Appendix 28 shows the significant difference between the average reward for both the decay methods, using 100 repetitions for reliable accuracy. In terms of average performance, exponential decay does provide a robust solution over multiple repetitions, however, due to the inconsistency of the linear decay method, I decided to test the consistency of exponential decay too. Appendices 29 and 30 show repeated trails of this algorithm over 10,000 steps with 100 different possible actions (machines), with 50 and 100 repeats. The two outputs showed a trend in inconsistency in the distribution of performance, both outputs, show positive outliers gapingly outperforming other trials, and roughly even distribution of average rewards between roughly 1.7 and 0.8, with additional negative outliers sparsely distributed between 0 and 0.8. This inconsistency highlights a huge flaw in the algorithm as there is a large and moderately consistent range of results, centred around the mean of 1.3 in this case. This makes the applicability of this algorithm plummet as consistency is important for implementing an algorithm into real world use. Arguably, the linear decay method may be more appealing for employment in the real world as it has a very dense range with exceptions to a predicable 5% insufficiency rate. Whilst this approach does provide strong theoretical promise, its inconsistency suggests unreliability which is essential to be robust for application in the real world and it cannot be relied upon at scale, despite being able to outperform other algorithms. From this hypothesis, I decided to explore a hybrid of this approach's theory, and the idea of the upper confidence bound, to refine and develop my artefact. In doing so I aim to adapt and improve these algorithms by removing future knowledge dependency for UCB and to remove the inconsistency in the exponential approach.

Hybrid

Following my previous approaches, with my upper confidence bound approaching proving to be the most efficient but less applicable, and my exponential approach having a strong theoretical foundation but not performing consistently. I decided to merge these two approaches, dedicating actions to randomly explore or to revert to a simplified upper confidence bound approach, without dependencies on knowing the number of spins inputted. This means that, total exploration is promoted offering a more reliable and constant approach at the sacrifice of initial numerical regret. This sacrifice is visible by the shallower initial gradient of the average rewards in comparison to the UCB and exponential approach (See Appendix 33). This output shows a slower determination of the optimal action; however, it is reaching it eventually unlike the exponential decay approach. Moreover, the largest problem with the exponential approach was its consistency. This is certainly not the case in this hybrid approach (See Appendix 34), where there is complete consistency in finding the optimal machine with only slight variations in the exploration phase. It also shows a consistent rate of exploration greater than even that of the upper confidence bounds (See Appendix 35). This slightly lower rate of exploration, coupled with a higher consistency than both UCB and exponential decay, provides conclusive evidence that is algorithm is the most applicable to real-world scenarios where extensive exploration is more suitable, and consistency is crucial. This was a significant discovery in my projects journey, concluding my solutions with an efficient, consistent and applicable approach as a product of refining the failures of all my previous approaches. Developing on the success of this hybrid solution, the following sections will explore the statistical and reinforcement learning theories that inform such algorithms and demonstrate their real-world application.

Theory

This section aims to provide further clarity and depth on the mathematics behind the problem, contributing to its significance. Additionally, aiming to provide clear information convincing the reader of the significance of the Multi-Armed Bandit. The initial topic of discussion (Gaussian Distribution), aims to provide the reader with a more robust understanding of the distribution of rewards returned from each machine, explaining why the use of this distribution function makes the problem applicable to real-world scenarios. The second topic of discussion (probability) aims to extend upon the first section by expressing the importance of understanding misconceptions of probability. By the use of a small-scale example, this section will unravel the source of perceived errors and limitations to my solutions to the MAB problem, as I express one of my most significant learning points I discovered in my project. The final topic (varying conditions) provides information on how my solutions to the problem, when translated from a hypothetical setting will need to adapt for changing variables in a more dynamic environment. This section provides a solid foundation for my next main section on application as it considers changes in the actions means, a common difference between the hypothetical problem and reality, using the example of changes in engagement in categories of videos on media platforms. Each section continues to express its relationship to the significance and solutions to MAB, show clarity and consistency in explanations and develop links to other sections of my dissertation.

Gaussian distribution

Gaussian distribution, otherwise known as normal or bell curve distribution, is a form of probability distribution which describes how values on a random variable are distributed.

Gaussian distribution has a great significance to almost every aspect of academic research, from Brownian Motion in physics to Intelligence Quotients in social sciences, even with many relationships and appearances in nature itself such as height of humans, due to the Central Limit Theorem, (Kwak and Kim, 2017). The Central Limit Theorem states that the sum of many independent random variables tend to follow a normal distribution, even if the original variables are not normally distributed. This is described as the most fundamental theory in modern statistics (Kwak and Kim, 2017). Taking the example of height of males in the UK, which is symmetrically distributed in a bell curve fashion of continuous data, where μ (mean height) is 175cm and σ (standard deviation) is around 7cm. Given any value of H (height) using a Gaussian probability density function (PDF), the percentage deviation can be worked out to find where males in the UK lie in comparison to others, (Chen, 2024), (See Appendices 39, 40 and 41). This relates to the Multi-Armed Bandit problem as the machine's biases are centralised around a Gaussian distribution with a means of 0 and a standard deviation of 1 (Murphy, 2007). Once each machine's means has been determined, rewards from that machine are distributed normally once more with the generated value as the means and a standard deviation of 1. This essentially creates a 3-dimensional bell curve (See Appendix 9), from where probabilities of rewards given any number of machines or spins can be derived. The graph shows a heat mapped 3-dimensional cone with 6 lines plotted to show the standard deviations, showing the regions to represent the 0.65, 0.95, 0.997 heuristic, (Kwak and Kim, 2017). This rule of thumb indicates that within ±σ of the means, 68% of the data points lie within these boundaries, 95% of the data points lie between ±2σ of the means and 0.997% of the data points lie between ±3σ of the means (see Appendix 36). Given this information, predictions based on the probability of means being generated given the number of machines can be an efficient way of estimating if a machine has reached the optimal action. For example, it can be estimated that if 666 machines are normally distributed, one of them is probable but not guaranteed to have a means =< +3σ

as there is a 0.15% chance of this occurring (666x0.15≈100%). Some approaches to the Multi-Armed Bandit problem use the number of machines to estimate the value of the highest machine, using that information then deciding if it is appropriate to exploit and to see if the reward return for a machine is a probable mean rather than a positive outlier. However, this approach is extremely reliant on formal probability, hence why I decided not to use it as a direct approach, but it has helped me understand the importance of probability as a cause of outliers in my programs. This leads me to the idea of 'luck' and the vast misinterpretations of probability (Crane, 2023).

Probability

If I were to generate a number between 1 and 100 inclusive, the probability of any number in that range being generated is 1/100. If one were to select a number in that range, intuitively, it might be expected that after generating 100 numbers, the number selected is extremely likely to appear, yet this may not be the case as each random generation event is independent from the last and therefore it is not guaranteed that the number will appear (Crane, 2023; Chen, 2024). This concept reflects the fundamentals of probability theory, which states that the expectation of an outcome does necessarily not equate to certainty (Kwak and Kim, 2017).

To directly test this, I created a small program (See Appendix 37) to generate 100 numbers between 1 and 100 and check how many numbers failed to appear over 100 trials, to find the average number of missing values, (See Appendix 38). The output showed after 100,000 trials, there was an average number of missing values at 36.522 which emphasises the uncertainty of probability models. This small test illustrates the 'danger' of relying solely on probability

expectations and highlights common misinterpretations, such as assuming uniform distribution will always lead to complete coverage (Murphy, 2007; Esfandiari et al., n.d.).

Although we are dealing with gaussian distribution in this problem rather than uniform random distribution, there is still that element of uncertainty described as the Gambler's Fallacy by (Crane, 2023). I decided to study into this to understand the basis of 'luck' in the world of mathematics as the Multi-Armed Bandit problem is based around bias and the interpretation of it as a gambling decision. Therefore, by reading into the problem further I discovered the importance of the methods in these algorithms in preventing falsely positive assumptions such as the confidence bounds acting as safety nets to keep testing exploration for the chance that 'luck' or probability was not 'on my side', as perhaps the distributed rewards were all represented below the actions means for the data collected so far (Li et al., 2010; Bandit Algorithms, 2016). I will develop on this idea that in real-world examples, depending on the application, these improbabilities or outliers are not to be cast aside as they can cause catastrophic failures with no room for error (Gamarnik and Li, 2018).

Varying Conditions

As explained earlier, the conditions I am working with in this Multi-Armed Bandit problem involve standard normal distribution N(0,1) with constant means and constant standard deviation (In my program it is referred to as variance). However, as Heraclitus stated, 'the only constant in life is change', meaning in the real world both means, and standard deviations change over time as a result of a dynamic environment. For example, as I previously mentioned,

Facebook and Google both use a variant of the UCB approach for advertising; however, unless the user had a dangerous addiction to one product and they were filled with excitement every time they saw an advert on that category for the rest of their lives, the algorithm would need to change. Using the example of advertisement (constantly promoted to me by a range of sources such as (Jahanbakhsh, 2020) and (Arzamasov, 2024)), I decided to research waves of interest that tended to be accurate in chronologically quantifying how an individual reacts to a category of social media content. Upon researching this I discovered the Gartner Hype Cycle from (Linden and Fenn, 2003) which was further elaborated to me by (Dedehayir and Steinert, 2016), highlighting the relationship a user has with media content over time, with an initially positive hype followed by a peak in engagement then a trough of disillusionment as it becomes too familiar and repetitive, then finally a slope of enlightenment and plateau of productivity, (See Appendix 46). I then quantified this wave using Desmos graph plotter where the total wave is stretched over the number of steps for each trial and applied to the means of each machine. As expected, the more naïve algorithms, with a constant rate of decreasing exploration, handled this varying condition poorly as it remained to exploit that machine regardless of its drop in performance dictated by the function I created. However, the hybrid approach showed greater adaptability to this varying condition due to its more extensive exploration phase. This highlights the importance of the dynamic balance between exploration and exploitation as the complexity of the problem increases when real-world conditions are applied. This extension of the original MAB problem, and the impacts it has on my algorithm, leads me to discuss how the problem may breach its hypothetical setting to the dynamic environment of the real-world.

Application

In this section, I am going to provide proof and explanations to the question of the significance of the Multi-Armed Bandit problem. Whilst I have only presented this problem as a hypothetical experiment with a contained environment of variables, this section aims to provide clear and relatable examples where the problem may be translated to fields such as social media, advertisement, trading and the simple example of an individual choosing what restaurant to go to. Another aim of this paragraph is to essentially fill that information gap between the technology and platforms we dependently use every day, and the logic and algorithms powering dictating them.

Restaurant Example

One the first representations of the problem that was proposed to me was using the example of choosing what restaurant to go to dinner for. This was a simple yet brilliant way from me to come to grips with the fundamentals of the problem introduced to me by (Ritvikmath, 2020). Referring to my initial and simple approach, ‘exploroit’, I am going to explain how I can use this solution and apply it to this example scenario. Consider an individual who migrates to a new town for a job. This town contains ten restaurants, each offering a different cuisine, none of which the individual is familiar with. They have no way of accessing information about these restaurants apart from trying them personally. The individual is working in the town for 100 days and decides to eat out every night. How will they gain the optimal cumulative pleasure return over this time from visiting these restaurants? The exploroit approach suggests visiting each restaurant for a fraction of the time and then exploiting the best restaurant for the

remaining duration, but it does not provide guidance on what ratio of exploration to exploitation to choose. For example, if the individual dedicates 30 days to visit each restaurant three times and spends the remaining 70 days exploiting the best option, this ratio may work for 100 days. However, if the individual decides to stay in the area for another 400 days, this initial ratio skews, increasing the probability of numerical regret if the wrong restaurant was chosen. The key issue is that in many cases, the number of days, steps, or spins is unknown to the individual, user, or bandit. As I mentioned previously, when scrolling through short videos on social media, users do not decide in advance how many times they will scroll or inform the application's algorithm before they start. This leads me to explain how my other solutions may be used in context of social media and advertisement.

Advertisement and Social Media

According to (Connell, 2024), 'In 2024, U.S. adult users were projected to spend a total of 4.8 billion minutes daily watching TikTok videos.'. Whilst this is perhaps a worrying figure, it serves the point of the efficiency of TikTok's ability to entice their viewers into spending an incredible amount of time on their platform, but why is this the case? Vague terms are often used to describe how TikTok is so efficient in displaying video's which will engage their viewers, 'The TikTok algorithm', 'Personalised recommendation system' or 'An attention economy', but what do all of this mean? All these terms loosely describe how TikTok and other social media platforms as able to tailor a user's experience to maximise their enjoyment or engagement from their platform, by use of a stratified algorithmic approach prompted by fundamentals of machine learning and data-driven decisions. This relates to the Multi-Armed Bandit problem, as the fundamentals of the task or problem at task are similar, maximising

rewards through exploitation and testing other actions via exploration. Let the bandit means of a machine translate to the means of the perceived enjoyment a user gets from a certain category of video, and let the reward be quantified by a mixture of videos in that category liked, commented on or re-watched. With the goal of the so called ‘TikTok algorithm’ to exploit these categories with high means, showing users familiar categories which are known to be enjoyed, whilst effectively balancing out exploration to find potential new categories the user may enjoy. For instance, if a user frequently watches fitness videos, TikTok may prioritize workout content (exploitation). However, if the user occasionally engages with self-improvement content, the algorithm might test this category by sporadically suggesting productivity videos (exploration). This example is supported by the likes of (Yan Y, 2024): ‘These algorithms adeptly navigate the trade-off between exploration and exploitation to maximize user engagement and satisfaction’. This demonstrated the relationship between how reinforcement techniques, initially designed for mathematical and theoretical problems, such as the Multi-Armed Bandit problem, can be directly applicable to the social media. The limitations of applying Bandit solutions to social media directly is the idea of varying conditions, as I mentioned under the varying conditions, due to factors such in preferences changing, lack of a direct reward with a range of information acting to determine the reward (time watched, likes, comments) which touches on the idea of missing data. For example, (Zhou et al., 2023) quotes that ‘Females tend to present significantly different emotional valence and social media content engagement behaviours compared to males’, which may lead to a lack of reward information for males meaning a lower enjoyment rate for them. Moreover, (Kal F, 2022) quotes "When comparing the performance of multi-armed bandit algorithms, the potential impact of missing data is often overlooked." Which shows the limitations of a direct application as data is complete within the theoretical environment of the Multi-Armed Bandit problem. However, an example where rewards are easily quantifiable is advertisement.

As I mentioned under the UCB subtitle, both Google and Facebook use the principles of an upper confidence bound to recommend adverts to users. As (Magon, 2019) states, *"Facebook Ads optimization using the Upper Confidence Bound (UCB) algorithm allows advertisers to dynamically allocate budget towards high-performing ads while still exploring new audience segments to enhance engagement."* This source demonstrates the relationship with the Multi-Armed Bandit problem and advertisement as it 'allocates budget towards high-performing ads' which is exploitation, 'while still exploring new audience segments to enhance engagement' which is exploration. The translation between our theoretical reward system is relatively simple as there is a Boolean (two state) description of interacting with the advert or not, making it explicitly obvious of the user's interest. The limitations or question of provenance to this application of the UCB algorithm is, what if someone mistakenly clicked on a pop-up advert? Well in the theoretical environment, the appearance of anomalies exists as an unprobeable reward distribution beyond 3σ, however this misinterpretation of the means is accounted for by the 'safety net' of the confidence bound, as the first initial values are not 'trusted' to be the means and after many values, one anomaly is not going to largely affect an average. The Multi-Armed Bandit problem underpins how social media platforms and digital advertisers optimise engagement by balancing exploitation of known user preferences with exploration of new content or ad placements. While MAB solutions work well in ad optimisation, challenges arise from missing data, user behaviour variability, and misinterpreted interactions. An area of application where MAB strategies which peaked my interests and are even more directly applicable, with clearly quantifiable rewards, is trading and investments, where algorithms must allocate capital efficiently while managing risk and uncertainty.

Trading

High-Frequency Trading (HFT) is the process of algorithmically placing large volumes of orders at extremely small timeframes, often milliseconds, in order to exploit very small price differences. This is an automated process dependent on very low-latency infrastructure, allowing trades to execute almost instantaneously to generate profit for a firm (Sweet, 2019). As Sweet (2019) highlights, "HFT relies on split-second decision-making processes that require advanced reinforcement learning models to optimize trade execution and minimize market impact."

This relates to the Multi-Armed Bandit problem, as the framework of the environment in trading algorithms is similar to the reinforcement learning used in MAB. For every share option, there are the finite actions of buy and sell, with a third state of hold (don't sell). This means that these algorithms can exploit the quantitative trends in trading by comparing the profit derived from actions on different stocks, encapsulating the trade-off between exploration (trying different strategies on new stocks) and exploitation (profiting on actions known to be successful) (Zhang, Zohren & Roberts, 2023). Zhang et al. (2023) explain that "MAB-based strategies are particularly useful in HFT as they dynamically adapt to changing market conditions, ensuring efficient capital allocation."

MAB models are foundational in machine learning; however, considering the conditions of trading, often rewards are delayed, and actions are not Boolean and don't directly translate to rewards. The reward is not only dependent on the action (e.g., buy this stock and get x reward) but also dependent on factors of time (when to buy the stock, e.g., troughs or peaks). (Vivanti, 2021) notes that "unlike classical MAB models, financial trading demands reinforcement

learning approaches that factor in delayed rewards, risk sensitivity, and volatility adjustments." Therefore, the use of Q-learning, a more advanced version of the reinforcement learning used in the Multi-Armed Bandit problem, is employed as it considers delayed rewards and market trends (Vivanti, 2021). While this requires a more complicated learning process, the fundamental methodology remains the same.

To demonstrate this, I programmed a Q-learning HFT model (see Appendix 18), using the Upper Confidence Bound (UCB) method to determine the reliability of each action. The model simulates historical stock price data 1,000 times to learn which actions are likely to return a reward. The algorithm uses this learned knowledge, stored in a Q-table (see Appendix 19), where each Q-value represents the expected reward for a given action at a specific state. Once trained, the algorithm prompts the user for:

A timeframe for investment (e.g., 1 minute).

The number of actions per second (defining trading frequency)

During live market execution, the algorithm:

Retrieves real-time stock data.

Uses the Q-table to determine optimal trading actions.

Executes trades based on this data.

Outputs each action, timestamp, investment amount, and money remaining.

Outputs the final profit or loss (see Appendix 20).

This model is one of my greatest successes in learning and building upon the idea of application of the Multi-Armed Bandit problem. As although this is only a model, it managed to make a profit on the live market from shares in Tesla, with the only limitation of applying this algorithm to the real market being the fact I don't have billions of dollars to invest into low-latency infrastructure and I am not a brokerage platform so I would be charged a brokerage fee for each trade, which would cause a net loss of profit as I am making thousands of trades to exploit margins as small of 1 cent.

This example is useful to demonstrate the significance of outliers in solutions to the Multi-Armed Bandit problem. Take a company that uses these algorithms for high-frequency trading, if they have a vast network of clients, but one of those clients loses all their money due to an outlier or improbability in the algorithm, or due to the algorithm's lack of responsiveness to market fluctuations, their other clients will become aware of this one huge failure and leave the company as it is too great of a risk. Sweet (2019) warns that "one catastrophic failure in an HFT algorithm can result in liquidity crises and investor panic, leading to significant systemic risks in financial markets."

From researching and developing my own example of application of the Multi-Armed Bandit problem in trading, I have learned the importance of consistency in my algorithms, to mitigate the risk of firms reliant on this technology from failing. Additionally, I have continued to explore the varieties of relevance and significance of MAB in the real world. "In the U.S. equity market, algorithmic trading accounts for approximately 60-73% of overall trading volume" (Benzinga, 2023). "Private investment and trading constitute about 16% of GDP" (Trading Economics, 2025). Together, these two statistics and my research proves the extreme significance of these algorithms, derived from reinforcement learning methods developed

through problems such as MAB, as they prove to be dependent factors to the US economy as a whole.

Conclusion

Solutions Conclusion

Having explained each solution, highlighting the individual successes and failures of each approach, to contextualise their performance I have created a spreadsheet comparing the results of each method:

Column1	Exponential decay	Linear decay	UCB	Hybrid
Total Numerical Regret	12151	14524	1731	3163
Average Action Regret	0.6606	1.3527	0.0634	0.1077
Persentage Regret	30.88%	63.23%	2.96%	5.03%
Inconsistancy %	6.40%	8.60%	7.20%	4.10%

In order to interpret these results, the glossary can be used to give an understanding of these terms. The first three rows are all ways of representing the amount of regret for each method, where the UCB method and the Hybrid method show dominancy with minimal regret. The second means of comparison was an inconsistency percentage, which shows the % of negative outliers which had an average reward 10% or more below the average of all trials. With these two factors, consistency and numerical performance, the most effective of my methods can be evaluated. It is clear that the Upper Confidence Bound method is the most effective shown by the minimal regret values, therefore being the most effective solution to the theoretical problem itself as it achieved the aim of pulling the optimal machine the most amount of times. However, by evaluating the inconsistency of these methods as a factor how applicable they are to the real world, the Hybrid approach proved the most effective with only 4.1% negative outliers whereas UCB had 7.2% negative outliers, this shows that the aim of the hybrid approach was effective as it took the more reliable exponential decay's consistency and the performance of the UCB method. By using both of these factors of efficiency into account, Appendix 44 shows that the Hybrid approach would be the best solution in a real world problem as the combination of the inconsistency and regret percentage was the lowest out of all my approaches, as it's consistency compensated for its slightly higher regret value, with a total negative performance of 9.13

compared to UCB's of 9.16. Appendix 45 displays the 'Total Numerical Regret' in table format to visualise their performance. In conclusion, the Upper Confidence Bound solution is the most effective solution to the Multi-Armed Bandit problem itself, and the Hybrid approach proves the most applicable to real world scenarios of machine learning using this model.

Application Conclusion

In this section of my dissertation, I have highlighted the significance of my artefact, 'The Multi-Armed Bandit Problem' by translating this hypothetical problem into real world scenarios such as social media, advertisement, trading and daily situations. I believe that my most compelling argument for the problem's significance is the section on high frequency trading as I created a working simulation of an automated trading algorithm which proved effective by generating $0.90 profit by taking 19 actions across a timeframe of 15 minutes, showing a 0.09% gain per 15 minutes, and a 0.36% gain per hour. Theoretically, if the market and the algorithm were to continue to perform at this rate for an entire year in live market hours, my profit would amount to roughly $46.92 trillion. This is direct proof for the significance of simple machine learning problems such as MAB as these algorithms form foundations for algorithms which shape economies and hugely influence the flow of money around the world.

Additionally, my research into how MAB forms foundations in algorithms for social media hold equal significance. These platforms are accessed by the majority of the world's population, displaying political information influencing voters' choices, promoting a variety of behaviours shaping the young populations morals and attitudes towards all areas of life and providing beliefs and cultures around the world from a spectrum of subjectiveness and perspectives. Regardless of censorship and government intervention with these sites, the algorithms created by the owners of these social media platforms hold the ultimate

responsibility of all information provided by these algorithms which influence and foster a plethora of understandings and beliefs, emphasising the great significance and impact these algorithms can hold. By the use of the example of TikTok and Google advertisement, this section demonstrated how the algorithms used in the Multi-Armed Bandit problem can be applied to recommend users content, in a clear and understandable fashion, providing the reader relatable examples to prove the significance of the problem.

In contrast to those large scale and distressing significances of the Multi-Armed Bandit problem, I researched and explained how the problem represents our daily decisions such as choosing from a variety of restaurants to eat at to 'what time should I do my homework today'? Often these decisions determined subconsciously, with little thought into how we arrive to the conclusion of our decisions, and the 'rewards' of these actions can be very abstract or qualitative but the problem of choosing the best action between many options still remains, where regret represents and opportunity cost of taking a suboptimal action. Whilst one wouldn't encode each decision into an algorithm to dictate their actions, an artificial intelligence may approach decisions such a way, hence why I decided to include this section in my dissertation. One of my main aims in this dissertation was to comprehensively and clearly demonstrate the significance of this problem to anyone, regardless of their prior knowledge of machine learning and programming, so this section effectively achieved this aim by explaining the translation of the simplicities of our lives into an algorithmic perspective.

Overall, this section provided robust evidence for the significance of the problem by use of simple and relatable examples such as social media to more complicated examples explaining how the fundamentals of Q-Learning can be derived from the problem to create automated trading algorithms. As I mentioned in the introduction to this section, my principle aims were

to provide clear, relevant explanations and to delve into the impacts of these algorithms to crucially foster the readers understanding of machine learning in order to 'bridge the gap' between individuals oblivious to the exponential growth of AI governance and the technology users blindly submit to and interact with every day. I have provided undeniable evidence of these objectives; however, limitations of my research were confined within a time scale and extent of relevance as I needed to focus on achieving the criteria required for the artefact EPQ, rather than deviating on a tangent providing extensive information without direct relevance to the title. Another limitation included a lack of publicly available information, such as the source code for TikTok's recommendation algorithm, so I could not directly compare my algorithms with TikTok's.

Theory Conclusion

In my theory section, I covered three topics, gaussian distribution, varying conditions and probability. The aims of this section were to not only provide context for the problem, introducing new information and depth to the problem, but also to provide further clarity on the relationship of this problem to real-world examples, convincing the reader of the significance of the problem. The first section of the topic, Gaussian distribution, describes the mathematics behind the distribution of the machines means and their respective rewards, clearly presented by a 3-Dimensional graph. Providing a clear explanation of 'means' and 'standard deviation' by the relatable example of height distributions in the UK. The section then developed on the 68, 95, 99.7 heuristic, describing the density of distribution of rewards on the probability density function (PDF) of a Gaussian distribution graph, containing self-written proof for the heuristic. Showing how probabilities can be derived from this knowledge to estimate the performance of machines. This allowed the reader to develop a robust understanding how such a mathematical background of my hypothetical problem can be

prevalent in the real world, achieving the other fundamental aim of this paragraph to delve into a deeper comprehension of the problem.

The section I covered under this topic was probability, describing the misconceptions associated with distributions of values. This section proved that logically perceived estimations often don't reflect reality, illustrated by a simple test I programmed. The aim of this section was to address my research in probability as often, appealingly unexpected results from my solutions occurred with seemingly slim probability, such as a large proportion of outliers. This section achieved the goals of applying further context to the problem whilst emphasising itself significance in real world scenarios through evidence such as the Gambler's Fallacy and how logical perceptions can drastically differ from real outcomes.

In my varying conditions section, I emphasised how the static conditions in the hypothetical setting of my problem often do not reflect real environments with dynamic variables. I used the example of changes in engagement in media content to explain how the mean return of rewards can change, meaning my algorithms need to adapt to it appropriately. This section achieved my aims by clearly explaining how my algorithms, when applied to real-world scenarios, will need to adapt to the specific situation's requirements, if knowledge is supplied of conditions varying from the hypothetical setting.

Evaluation

Extent of Success

Looking back at my introduction and plan, I have achieved my primary goals of creating efficient solutions to the problem and explaining this problems relationship with the real world and the development of technology. I set out to provide a number of efficient solutions to satisfy both the hypothetical problem and extensions of the problem into real-world scenarios and provide proof for the significance of the problem by creating algorithms and delivering research. From the simplicity of ‘exploroit’ to the complexities of my UCB and exponential decay hybrid approach, I can confidently remark that all my time has proven worthwhile as I have been following a trajectory of exciting education and understanding. When I initially decided upon my EPQ title and began to create a plan, my expectations for the success of this project were undeniably underestimated as I could not have imagined the extent of success’s I have achieved. For example, my passion for the project led me to extend my title to develop on the significance of the problem, allowing me to expand my knowledge in mathematics, create an automatic trading algorithm and build upon my understanding of technology as a whole. I would go as far to say this project has sparked my passion in machine learning so much that I have goals to continue to thrive in developing my knowledge in the field with a dream to pursue this profession.

Limitations

Due to the passion for this project, I would desire to continue to write, research and create more content limitlessly, however time constraints provided a difficult limitation of this. One of my greatest struggles in creating this project in a limited time frame, was deterring myself from going on tangents, providing information deviating too far from the title and the requirements of the EPQ process. Besides that, I felt comfortable in managing my time, creating personal deadlines for different sections in my main discussion and meeting them comfortably, until this flow was disrupted by an unexpected illness which disabled me from working on my project for two weeks, right before the final deadline. I manged this limitation by requesting an extension ahead of time, giving me the opportunity to catch up and not rush my final sections. Another limitation arose from changing my entire EPQ five weeks into the process, as I was initially going to discuss 'to what extent is commercial quantum computation achievable within a decade'. This change happened a couple days before my Literature Review draft was due in, which would have been very difficult to manage, but again I was granted a small extension to catch up, and I assigned time over the Christmas holiday to catch up on areas I was lacking in. In hindsight, I am very thankful I made this change due to Microsoft's release of 'Majorana 1', the world's first quantum computer, only a few days before the EPQ deadline, which would have caused me difficulties as it would have completely invalidated my project.

I believe the most prevalent of my difficulties was planning and structuring my project, due to its unorthodox nature of creating an artefact in code. From the help of my EPQ teacher, I was able to manage this difficulty by expressing the artefact in my Appendix, whilst following the same three section structure of a dissertation.

Research Process

Due to the duality of my project, I had to research materials which would aid me in the programming aspect and researching significance of the problem. I had little difficulties in finding efficient materials and sources of various natures to support my project, due to the current relevance and popularity of the problem. Despite talking about social media rather a lot in this project, I did not use it as a source of materials or information directly; however, my programming ability has been developed through tutorials on social media, prior to starting my EPQ. As programming is common practice in the modern world, there are a plethora of helpful platforms available to learn practices on and query specific questions. In the rare occasion these platforms were not sufficient in helping me, my Computer Science teacher at school proved a huge asset to my creation of my artefact as he was always willing and able to provide very helpful feedback. From researching so many sources and materials for my EPQ, I have learned the importance of personal opinion rather than reliance on the information of one source, as misinformation tended to reveal itself as sources contradicted and disproved one another. This takeaway, alongside an increased proficiency in finding effective sources quickly, as valuable skills I have derived from the EPQ process, to which I would have otherwise been oblivious.

Future goals

In the future, I aim to expand more upon the relationship of my algorithms to fundamentals of reinforcement learning and Q learning. I have created a few other approaches including Boltzmann's Distribution, SoftMax and Temperature decay, which I would like to do a write-up on as I was limited by time so I could not include them in my EPQ. This project has amplified my passion for programming, the study of machine learning and algorithmic trading.

These are areas which I would like to explore as a profession in the future, and the EPQ has provided me a brilliant opportunity to express my passion for the topic for employment. Overall, I am extremely grateful for the opportunity to partake in the EPQ experience, as it has not only developed my passion and interest in the topic, but allowed me to spend time practicing a hobby I enjoy with a rewarded grade at the end of the experience. I have learned essential skills I will carry on with me for life, from researching, structuring and writing long dissertations, and logging progress for a project in ProjectQ, which are exclusive to the EPQ as an academic experience this far through education.

Bibliography

Aibin, Micheal. baeldung (2020). Baeldung. [online] Baeldung on Computer Science. Available at: https://www.baeldung.com/cs/logarithmic-time-complexity?utm_source=chatgpt.com [Accessed 19 Jan. 2025].

Auer, P., Cesa-Bianchi, N. and Fischer, P. (2002) 'Finite-time analysis of the multiarmed bandit problem', Machine Learning, 47(2-3), pp. 235–256. doi: 10.1023/A:1013689704352.

Arzamasov, V., 2024. *Optimizing Marketing Campaigns with Budgeted Multi-Armed Bandits.* Towards Data Science, 16 August. [online] Available at: https://towardsdatascience.com/optimizing-marketing-campaigns-with-budgeted-multi-armed-bandits-a65fccd61878 [Accessed 28 February 2025].

Bandit Algorithms. (2016). The Upper Confidence Bound Algorithm. [online] Available at: https://banditalgs.com/2016/09/18/the-upper-confidence-bound-algorithm/?utm_source=chatgpt.com [Accessed 19 Jan. 2025].

Benzinga. (2023). What Percentage of Stock Trades Are Made By Bots And Algorithms? [online] Available at: https://www.benzinga.com/general/topics/23/06/32861724/what-percentage-of-stock-trades-are-made-by-bots-and-algorithms [Accessed 9 Feb. 2025].

Chen, James. “Normal Distribution.” Investopedia, 13 Mar. 2024, www.investopedia.com/terms/n/normaldistribution.asp.

Cohn, Henry, Wikipedia. (2020). Computational complexity of mathematical operations. [online] Available at: https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations.

Connell, A. (2024). 29 Top Facebook Video Statistics For 2025. [online] Adam Connell. Available at: https://adamconnell.me/facebook-video-statistics/?utm_source=chatgpt.com [Accessed 7 Feb. 2025].

Crane, A. (2023). The Mathematics of Luck: Understanding Randomness and Probability. [online] Untamed Science. Available at: https://untamedscience.com/blog/the-mathematics-of-luck-understanding-randomness-and-probability/[Accessed 2 Feb. 2025].

Dedehayir, O. and Steinert, M. (2016). The Hype Cycle model: a Review and Future Directions. Technological Forecasting and Social Change, 108(108), pp.28–41. doi:https://doi.org/10.1016/j.techfore.2016.04.005.

DeepMind (2020). AlphaGo - The Movie | Full Documentary. YouTube. Available at: https://www.youtube.com/watch?v=WXuK6gekU1Y.

Dean, J. (2017). How Will Artificial Intelligence Affect Your Life . Jeff Dean . TEDxLA. YouTube. Available at: https://www.youtube.com/watch?v=BfDQNrVphLQ.

Esfandiari, H., Research, G., Mirrokni, V. and Schneider, J. (n.d.). Anonymous Bandits for Multi-User Systems. [online] Available at: https://proceedings.neurips.cc/paper_files/paper/2022/file/50a057e9fe79ffa3f4120fb6fb88071a-Paper-Conference.pdf [Accessed 23 Jan. 2025].

Gamarnik, D. and Li, Q. (2018). Finding a large submatrix of a Gaussian random matrix. The Annals of Statistics, 46(6A). doi: https://doi.org/10.1214/17-aos1628.

GridflowAI (2023). Part 2: In-depth Exploration on Epsilon-greedy algorithm. [online] Medium. Available at: https://medium.com/@gridflowai/part-2-in-depth-exploration-on-epsilon-greedy-algorithm-2b19e59bbe22.

Han, S. and Kwak, I.-Y. (2023). Mastering data visualization with Python: practical tips for researchers. Journal of minimally invasive surgery, [online] 26(4), pp.167–175. doi:https://doi.org/10.7602/jmis.2023.26.4.167.

Heidi B. Clark, James Egger & Vincent G. Duffy and Mesko, B. (2023). A Review of Technology Giants' Healthcare Collaborations. MHealth, 9, pp.17–17. doi:https://doi.org/10.21037/mhealth-22-45.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P. and Gérard-Marchant, P. (2020). Array Programming with NumPy. Nature, 585(7825), pp.357–362. doi:https://doi.org/10.1038/s41586-020-2649-2.

Jahanbakhsh, K. (2020). Applying Multi-armed Bandit Algorithms to Computational Advertising. [online] arXiv. Available at: https://arxiv.org/abs/2011.10919 [Accessed 28 February 2025].

Kal F. (2022). Some Performance Considerations When Using Multi-Armed Bandit Algorithms. BMC Medical Research Methodology, 22(1), 1-12.

Kwak, S.G. and Kim, J.H. (2017). Central Limit theorem: the Cornerstone of Modern Statistics. Korean Journal of Anesthesiology, [online] 70(2), pp.144–156. doi:https://doi.org/10.4097/kjae.2017.70.2.144.

Lattimore, T. and Szepesvári, C. (2020). Bandit algorithms. Cambridge: Cambridge University Press. doi: 10.1017/9781108571401.

Lee, H. (2023). Multi-Armed Bandit and UCB Algorithm. [online] Harin Lee. Available at: https://harinboy.github.io/posts/UCB-Algorithm/?utm_source=chatgpt.com [Accessed 19 Jan. 2025].

Li, Lihong, et al. A Contextual-Bandit Approach to Personalised News Article Recommendation. Google Scholar. http://rob.schapire.net/papers/www10.pdf. 26 Apr. 2010.

Linden, A. and Fenn, J. (2003). Gartner Understanding Gartner's Hype Cycles. [online] Available at: http://ask-force.org/web/Discourse/Linden-HypeCycle-2003.pdf.

Magon, V. (2019). FB-Ads-Opt-UCB: The easiest way to optimize Facebook Ads using Upper Confidence Bound Algorithm. GitHub Repository. Available at: https://github.com/vaibhavmagon/FB-Ads-Opt-UCB (Accessed: 7 February 2025).

Moorfields.nhs.uk. (2016). Google DeepMind - Moorfields Eye Hospital. [online] Available at: https://www.moorfields.nhs.uk/research/google-deepmind.

Murphy, K. (2007). Conjugate Bayesian analysis of the Gaussian distribution. Available at: Google Scholar.

ritvikmath (2020). Multi-Armed Bandit : Data Science Concepts. [online] YouTube. Available at: https://www.youtube.com/watch?v=e3L4VocZnnQ [Accessed 4 Feb. 2025].

Stack Exchange. “Choosing and Designing Decay Types for Epsilon-Greedy Exploration in Reinforcement Learning.” Artificial Intelligence Stack Exchange, 3 Apr. 2023, ai.stackexchange.com/questions/39896/choosing-and-designing-decay-types-for-epsilon-greedy-exploration-in-reinforceme. Accessed 12 Nov. 2024.

“Teaching.” David Silver, 2015, www.davidsilver.uk/teaching/.

TED, K.S. (2011). How algorithms shape our world | Kevin Slavin. YouTube. Available at: https://www.youtube.com/watch?v=TDaFwnOiKVE.

Tegmark, M. (2017). Life 3.0: Being Human in the Age of Artificial Intelligence. London: Penguin Books, pp.85–133.

Tegmark, M. (2017b). Life 3.0: Being Human in the Age of Artificial Intelligence. London: Penguin Books, pp.1–21.

Trading Economics. (2025). United States GDP Growth Rate. [online] Available at: https://tradingeconomics.com/united-states/gdp-growth [Accessed 9 Feb. 2025].

Vermorel, Joannès, and Mehryar Mohri. “Multi-Armed Bandit Algorithms and Empirical Evaluation.” Machine Learning: ECML 2005, 2005, pp. 437–448, https://doi.org/10.1007/11564096_42.

Visual Studio Code. (2024). Version 1.75. Microsoft Corporation. Available at: https://code.visualstudio.com/.

Vivanti, R. (2021). Can Q-learning solve Multi-Armed Bandits? ResearchGate. Available at: https://www.researchgate.net/publication/355495717_Can_Q-learning_solve_Multi_Armed_Bantids [Accessed 9 Feb. 2025].

Weng, Lilian. “Exploration Strategies in Deep Reinforcement Learning.” Lilianweng.github.io, 7 June 2020, lilianweng.github.io/posts/2020-06-07-exploration-drl/.

WPSU Creative Services (2021). Coded Bias . Vimeo. Available at: https://vimeo.com/504432040.

www.youtube.com. (2023). Can AI Catch What Doctors Miss? . Eric Topol . TED. [online] Available at: https://www.youtube.com/watch?v=ll5LY7wI_Xc.

Yan, Y. (2024). The Evolution and Impact of Multi-Armed Bandit Algorithms in Social Media. Applied and Computational Engineering, 68(1), 150-158.

Zhou, Y., Li, X., Wang, H., & Chen, J. (2023) ‘Gender differences in emotional valence and engagement with social media content’, Frontiers in Psychology, 14, p. 9855065. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC9855065/ (Accessed: 7 February 2025).

Glossary

Action regret:

Pulling any machine that does not have the highest means. Each suboptimal action increases the action regret count by one.

Action/pull/spin:

These words ae used interchangeably in order to describe acting on the choice of a machine in order to return a reward. Where 'pull' and 'spin' are appropriate terms derived from one of the Bandit's arms spinning or pulling the machine.

Algorithm:

A step-by-step procedure or set of rules followed by a program in order to perform a task. In the context of the Multi-Armed Bandit problem, this refers to the encoded instructions determining the choice between machines for each spin.

Bandit:

The hypothetical being which decides which actions to take.

Contextual Bandits:

A variant of the Multi-Armed Bandit problem where additional 'context' or features are used to guide decision making, rather than solely one returned reward. This term appears when discussing extensions of the problem such as in social media where the decision-making process is based upon a multitude of 'rewards' (comments, likes, shares, etc.).

Epsilon Value ($\boldsymbol{\varepsilon}$):

This is the value used to determine whether to explore or exploit, when compared against a random variable. This value may be constant or decay over time to promote exploitation.

Exploration:

Trying different machines to gather more data to estimate their mean reward (bias).

Exploitation:

Choosing the machine that is currently believed to offer the highest reward.

Exploration Decay:

Exploration decay in context of MAB, is the method by which the algorithm gradually reduces its exploration rate over time (typically applied to an epsilon value for application).

Gaussian (Normal) Distribution:

A distribution model which takes weighted probabilities for continuous values from a bell curve shaped probability density function (PDF). This function is represented by this equation:

$f(x) = (1 / (\sigma\sqrt{(2\pi)})) \bullet \exp(-((x - \mu)^2 / (2\sigma^2)))$

μ = mean value

σ = standard deviation

Mean (μ):

The value where the distribution for the rewards in each machine or action is centred around.

Machines:

In the context of the problem, the arms of the bandit are connected to machines which represent the different choices or actions available to the algorithm. Each machine has an unknown differing means and a known homogeneous standard deviation.

Q-Learning:

A type of machine learning that derives its actions from a value function applied to collected data, similar to the methods of MAB.

Regret:

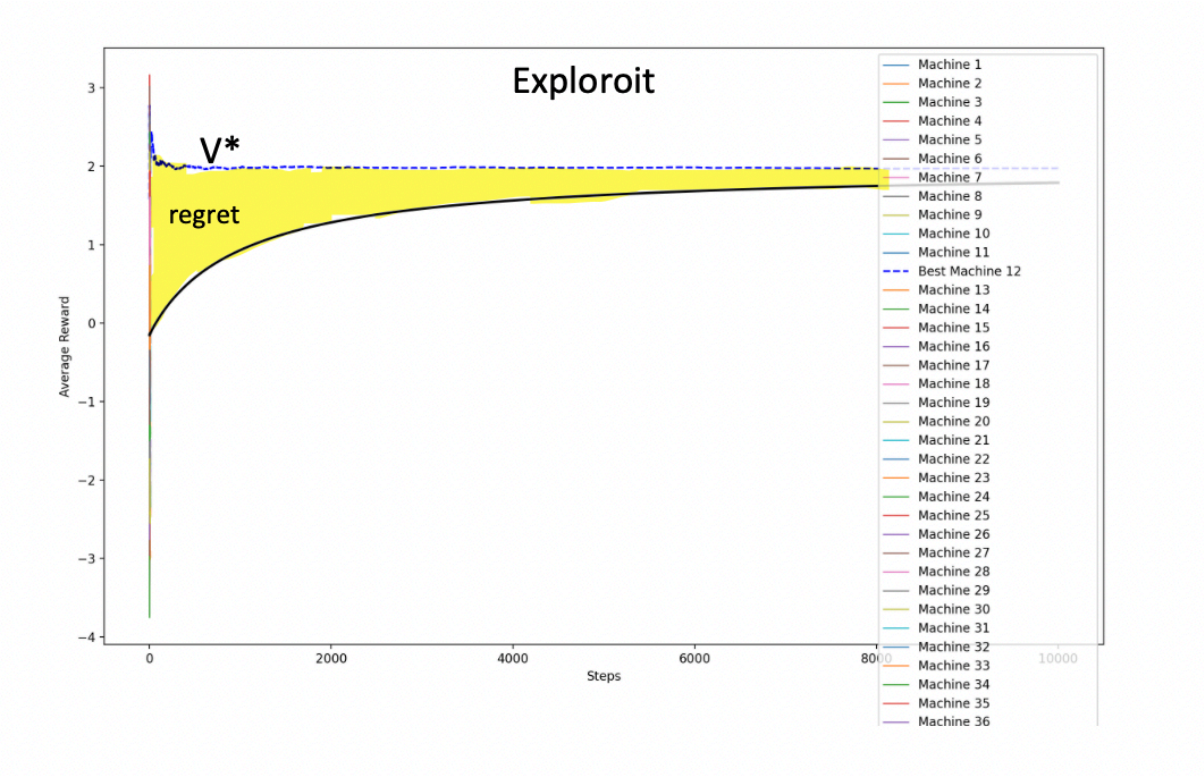
The difference between the reward returned from pulling a chosen machine (taking an action) and the reward return from pulling the optimal machine (the best action). Quantifying the opportunity cost of suboptimal decisions.

Standard Deviation (σ):

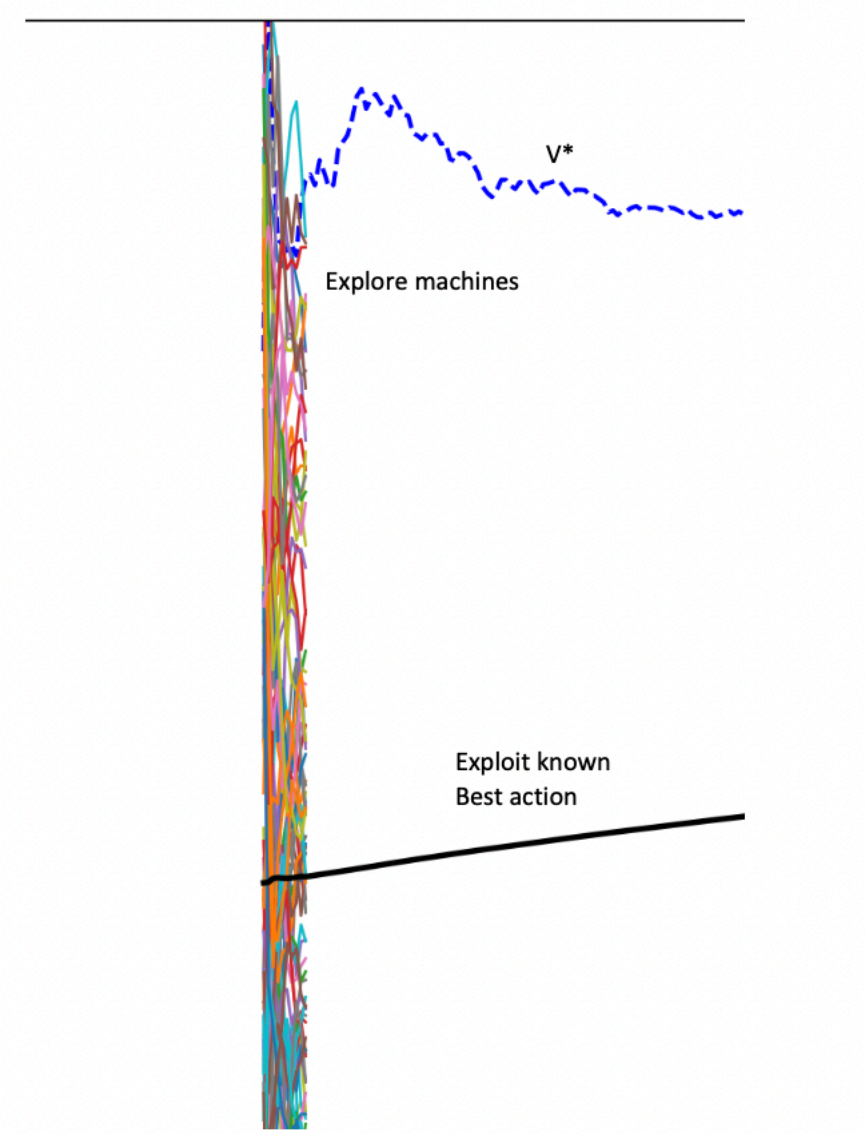
A value which dictates the variability and the spread of the distribution of rewards. This is referred to as variance in my programs.

Appendix

Appendix 1



Appendix 2



Appendix 3

$$\text{constant} = \sqrt{\frac{2\ln(\text{spins})}{\text{machines}}}$$

The **Upper Confidence Bound (UCB)** formula becomes:

$$UCB_i = \bar{x}_i + \sqrt{\frac{2\ln(\text{spins})}{\text{machines}}} \cdot \sqrt{\frac{2\ln t}{n_i}}$$

Where:

1. $\bar{x}_i$: The average reward for the i-th machine:

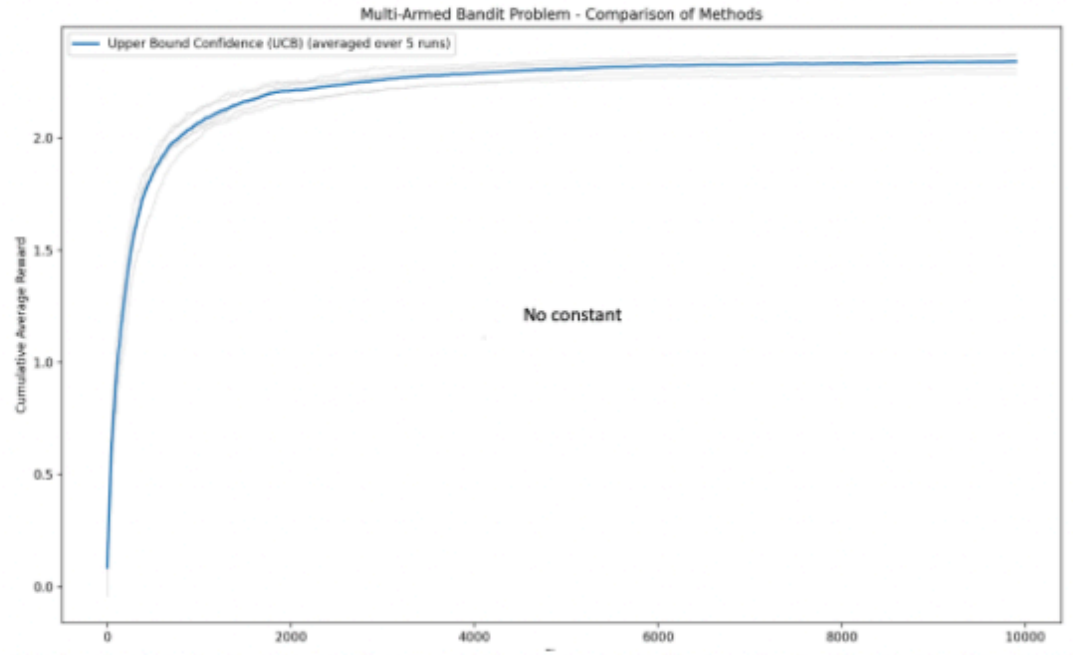
$$\bar{x}_i = \frac{\text{Total reward from machine } i}{\text{Number of times machine } i \text{ was selected}}$$

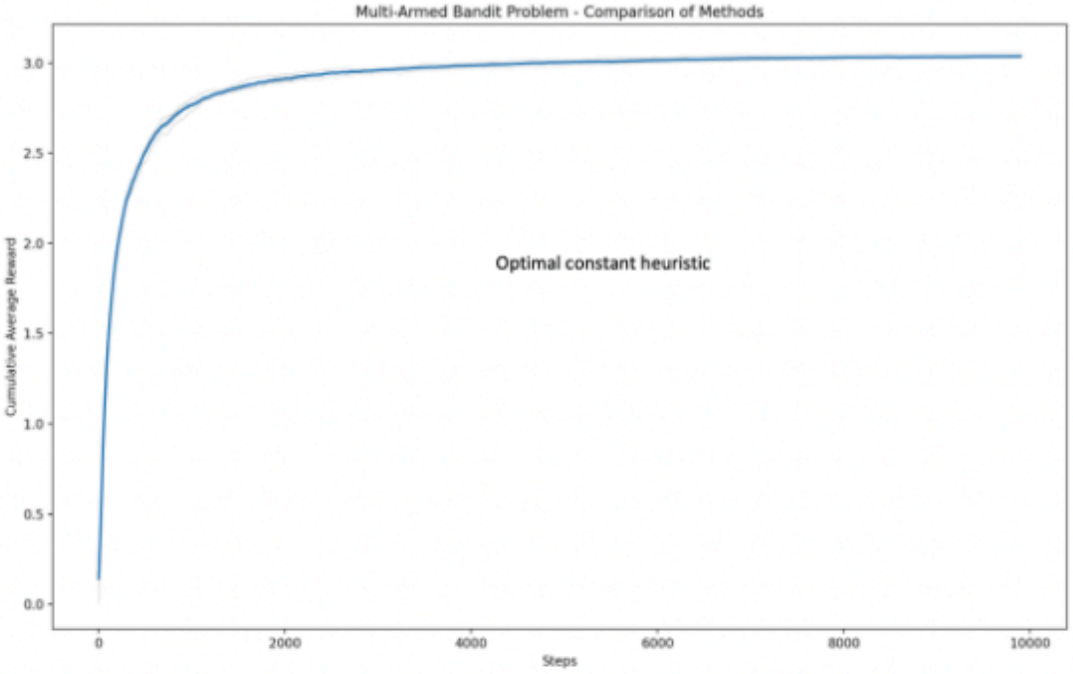
2. $\ln(\text{spins})$: The natural logarithm of the total number of spins (used in the constant).
3. $\ln t$: The natural logarithm of the current step t.
4. n_i: The number of times machine i has been selected so far.

This means the exploration term combines two sources of logarithmic growth:

- The total number of spins ($\ln(\text{spins})$) factored into the constant.
- The specific step in the process ($\ln t$) for adaptive exploration.

Appendix 4


Multi-Armed Bandit Problem - Comparison of Methods
Upper Bound Confidence (UCB) (averaged over 5 runs)
Cumulative Average Reward
No constant
0.0
0.5
1.0
1.5
2.0
0
2000
4000
6000
8000
10000


Multi-Armed Bandit Problem - Comparison of Methods
Cumulative Average Reward
Optimal constant heuristic
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
10000
Steps

Appendix 5

Combinedmulti.py script

```
import matplotlib.pyplot as plt
import numpy as np
import math
import random

def pad_rewards(rewards, target_length):
    return rewards + [float('nan')] * (target_length - len(rewards))

def exponential_decay(machines, spins, BanditMeans):
    epsilon_0 = 1.0
    decay_rate = 0.01
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        epsilon = epsilon_0 * math.exp(-decay_rate * i)
        if random.random() > epsilon:
            selected_machine = np.argmax(rewards)
        else:
            selected_machine = random.randint(0, len(rewards) - 1)

        reward = np.random.normal(BanditMeans[selected_machine], 1)
        rewards[selected_machine] += reward
        pulls[selected_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
```

```
        acr.append(cumulative_avg_reward)

    return acr

def linear_decay(machines, spins, BanditMeans):
    epsilon_0 = 1.0
    decay_rate = 0.01
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        epsilon = epsilon_0 * (decay_rate / (i + 1))
        if random.random() > epsilon:
            chosen_machine = np.argmax(rewards)
        else:
            chosen_machine = random.randint(0, len(rewards) - 1)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr

def ucb_method(machines, spins, BanditMeans):
    constant = math.sqrt(2 * math.log(spins) / machines)
    rewards = [0.0] * machines
    counts = [0] * machines
    acr = []
```

```
    for i in range(machines):
        reward = np.random.normal(BanditMeans[i], 1)
        rewards[i] += reward
        counts[i] += 1

    for j in range(machines, spins):
        upper_bound_all = []
        for i in range(machines):
            avg_reward = rewards[i] / counts[i]
            exploration_term = constant * math.sqrt((2 * math.log(j + 1)) /
counts[i])
            upper_bound = avg_reward + exploration_term
            upper_bound_all.append(upper_bound)

        selected_machine = np.argmax(upper_bound_all)
        reward = np.random.normal(BanditMeans[selected_machine], 1)



        rewards[selected_machine] += reward
        counts[selected_machine] += 1

        avg_reward = sum(rewards) / sum(counts)
        acr.append(avg_reward)

    return acr

def epsilon_greedy(machines, spins, BanditMeans, epsilon=10):
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0
```

```
    for i in range(spins):
        if i % epsilon != 0:
            chosen_machine = np.argmax(rewards)
        else:
            chosen_machine = random.randint(0, len(rewards) - 1)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr

def softmax_decay(machines, spins, BanditMeans, T0=1.0):
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        T = T0 * np.exp(-i / spins)

        avg_rewards = [rewards[j] / pulls[j] if pulls[j] > 0 else 0 for j in
range(machines)]
        exp_values = np.exp(np.array(avg_rewards) / T)
        probabilities = exp_values / np.sum(exp_values)

        chosen_machine = np.random.choice(machines, p=probabilities)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
```

```
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr


def main():
    machines = int(input("Enter the number of machines: "))
    spins = int(input("Enter the number of spins: "))
    repeats = int(input("Enter the number of repetitions: "))
    variance=1
    BanditMeans = [np.random.normal(0, variance) for _ in range(machines)]
    methods = {
        '1': ("Exponential Decay", exponential_decay),
        '2': ("Linear Decay", linear_decay),
        '3': ("Upper Bound Confidence (UCB)", ucb_method),
        '4': ("Epsilon-Greedy", epsilon_greedy),
        '5': ("SoftMax with Temperature Decay", softmax_decay)
    }

    print("Select methods to run (separate by commas for multiple):")
    for key, (name, _) in methods.items():
        print(f"{key}: {name}")

    selected_methods = input("Your choice: ").split(',')
    plt.figure(figsize=(12, 8))
    plt.xlabel('Steps')
    plt.ylabel('Cumulative Average Reward')
    for method_key in selected_methods:
        if method_key.strip() in methods:
            method_name, method_func = methods[method_key.strip()]
```

```
            all_runs = []
            cumulative_rewards = np.zeros(spins)

            for _ in range(repeats):
                rewards = method_func(machines, spins, BanditMeans)
                rewards = pad_rewards(rewards, spins)
                all_runs.append(rewards)
                cumulative_rewards += np.nan_to_num(rewards)

                plt.plot(rewards, color='lightgray', linewidth=0.8, alpha=0.7)

            cumulative_rewards /= repeats
            cumulative_rewards = np.where(cumulative_rewards == 0, float('nan'),
cumulative_rewards)
            plt.plot(cumulative_rewards, label=f"{method_name} (averaged over
{repeats} runs)", linewidth=2)

    plt.title("Multi-Armed Bandit Problem - Comparison of Methods")
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

Appendix 6

Exploroit script

```
import matplotlib.pyplot as plt
import numpy as np
import math


plt.figure(figsize=(20, 16))
plt.xlabel('Steps')
plt.ylabel('Average Reward')


variance = int(input('variance - '))
machines = int(input('machines - '))
practice = int(input('practice spins - '))
spins = int(input('spins - '))


BanditMeans = [np.random.normal(0, 1) for _ in range(machines)]

all_averages = []
best_machine_avg = 0
best_machine = 0
best_avg = []
```

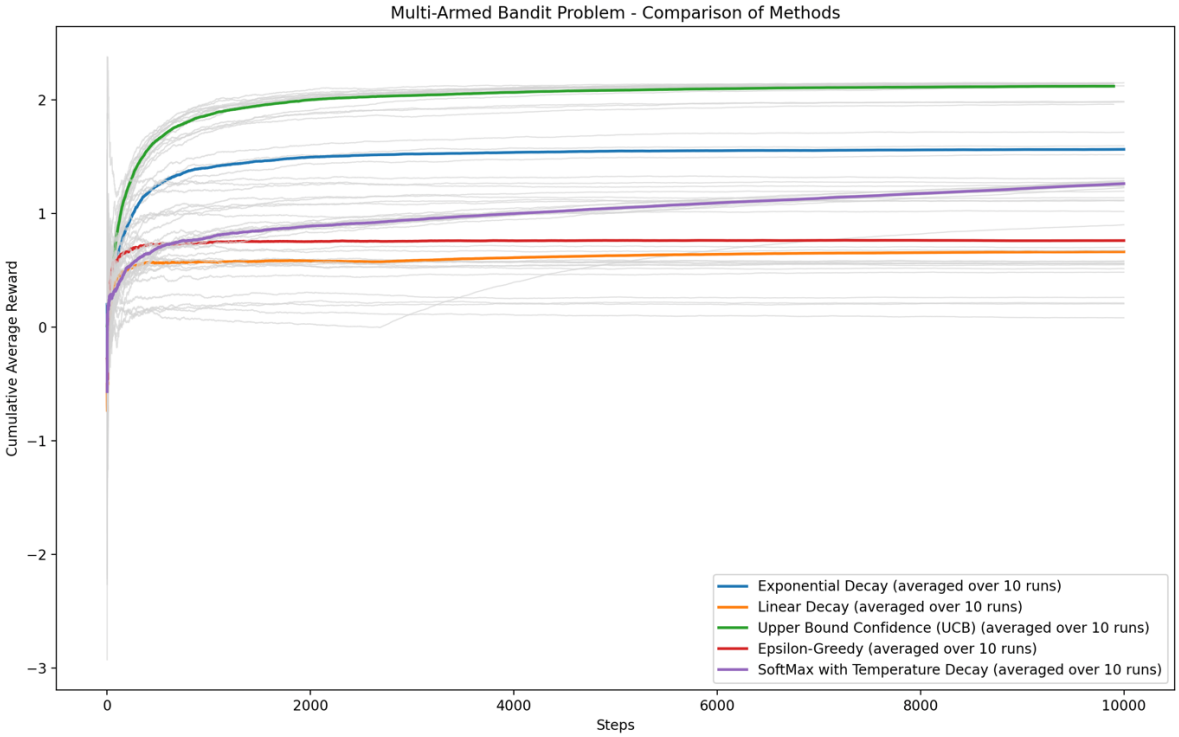
```
#could be better improved to find optimum ratio of spins to practive for x machines
for j in range(machines):
    banditr = []
    averages = []
    for i in range(practice):
        reward = np.random.normal(BanditMeans[j], variance)
        banditr.append(reward)
        avg = sum(banditr) / len(banditr)
        averages.append(avg)


    if float(averages[-1]) > best_machine_avg:
        best_machine_avg = averages[-1]
        best_machine = j
        best_avg = averages.copy()
    all_averages.append(averages)

count = 0

#with varying means
for i in range(spins - practice):
    count+=1
    reward = np.random.normal(BanditMeans[best_machine], variance)
    BanditMeans[best_machine]+= (math.sin((0.1*count)))

    best_avg.append((best_avg[-1] * len(best_avg) + reward) / (len(best_avg) + 1))


all_averages[best_machine] = best_avg


max_length = max(len(avg) for avg in all_averages)
all_averages_padded = []

for avg in all_averages:
```

```
    avg += [np.nan] * (max_length - len(avg))  #
    all_averages_padded.append(avg)


all_averages_padded = np.array(all_averages_padded, dtype=float)


overall_cumulative_avg = []


cumulative_total = 0
cumulative_count = 0

for i in range(max_length):
    if i < practice:

        valid_values = [all_averages_padded[j][i] for j in range(machines)]
        valid_values = [v for v in valid_values if not np.isnan(v)]
        cumulative_total += sum(valid_values)
        cumulative_count += len(valid_values)
    else:
        valid_value = all_averages_padded[best_machine][i]
        if not np.isnan(valid_value):
            cumulative_total += valid_value
            cumulative_count += 1


    overall_cumulative_avg.append(cumulative_total / cumulative_count)


for j in range(machines):
    if j != best_machine:
        plt.plot(all_averages_padded[j][:practice],   label=f'Machine   {j   +   1}',
linewidth=1)
    else:
```

```
        plt.plot(all_averages_padded[j], label=f'Best Machine {j + 1}', linestyle='-
-', color='blue')



plt.plot(overall_cumulative_avg, color='black', label='Overall Cumulative Average',
linewidth=2)


plt.legend()
plt.show()
```

Appendix 7

```
Enter the number of machines: 100
Enter the number of spins: 10000
Enter the number of repetitions (seperated by comma): 5
Select methods to run (separate by commas for multiple):
1: Exponential Decay
2: Linear Decay
3: Upper Bound Confidence (UCB)
4: Epsilon-Greedy
5: SoftMax with Temperature Decay
Your choice: 1,2,3,4,5
```



Appendix 8

3D gaussian Distribution Script, normaldistribution.py

```
import matplotlib.pyplot as plt
import numpy as np


# Parameters
mu_x, mu_z = 0, 0  # Means
sigma = 1  # Standard deviation
extent = 5  # Range for x and z


# Create a grid of x and z values
x = np.linspace(-extent, extent, 500)
z = np.linspace(-extent, extent, 500)
```

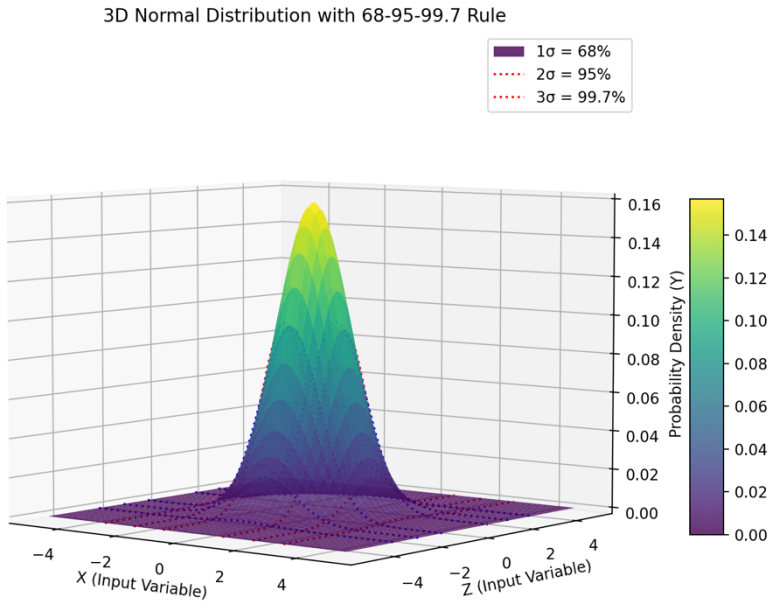
```
X, Z = np.meshgrid(x, z)

# Calculate the 3D Gaussian function
Y = (1 / (2 * np.pi * sigma**2)) * np.exp(-((X - mu_x)**2 + (Z - mu_z)**2) / (2 *
sigma**2))

# Plotting the 3D surface
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Z, Y, cmap='viridis', edgecolor='none', alpha=0.8)

# Add lines for the 68-95-99.7 rule on x and z axes
for i in range(1, 4):  # 1, 2, and 3 standard deviations
    # Lines for the x-axis
    x_line = np.full_like(z, mu_x - i * sigma)  # Create an array with the same shape
as z
    z_line = z  # Use the existing z array
    y_line = (1 / (2 * np.pi * sigma**2)) * np.exp(-((x_line - mu_x)**2 + (z_line -
mu_z)**2) / (2 * sigma**2))
    ax.plot(x_line, z_line, y_line, color='red', linestyle='dotted', linewidth=1.5,
label=f'{i}σ' if i == 1 else None)

    x_line = np.full_like(z, mu_x + i * sigma)  # Create an array with the same shape
as z
    y_line = (1 / (2 * np.pi * sigma**2)) * np.exp(-((x_line - mu_x)**2 + (z_line -
mu_z)**2) / (2 * sigma**2))
    ax.plot(x_line, z_line, y_line, color='red', linestyle='dotted', linewidth=1.5)

    # Lines for the z-axis
    z_line = np.full_like(x, mu_z - i * sigma)  # Create an array with the same shape
as x
    x_line = x  # Use the existing x array
```

```
    y_line = (1 / (2 * np.pi * sigma**2)) * np.exp(-((x_line - mu_x)**2 + (z_line - mu_z)**2) / (2 * sigma**2))
    ax.plot(x_line, z_line, y_line, color='blue', linestyle='dotted', linewidth=1.5, label=f'{i}σ' if i == 1 else None)

    z_line = np.full_like(x, mu_z + i * sigma)  # Create an array with the same shape as x
    y_line = (1 / (2 * np.pi * sigma**2)) * np.exp(-((x_line - mu_x)**2 + (z_line - mu_z)**2) / (2 * sigma**2))
    ax.plot(x_line, z_line, y_line, color='blue', linestyle='dotted', linewidth=1.5)



ax.set_title("3D Normal Distribution with 68-95-99.7 Rule")
ax.set_xlabel("X (Input Variable)")
ax.set_ylabel("Z (Input Variable)")
ax.set_zlabel("Probability Density (Y)")


ax.legend(["1σ = 68%", "2σ = 95%", "3σ = 99.7%"], loc="upper right")


fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)

plt.show()
```

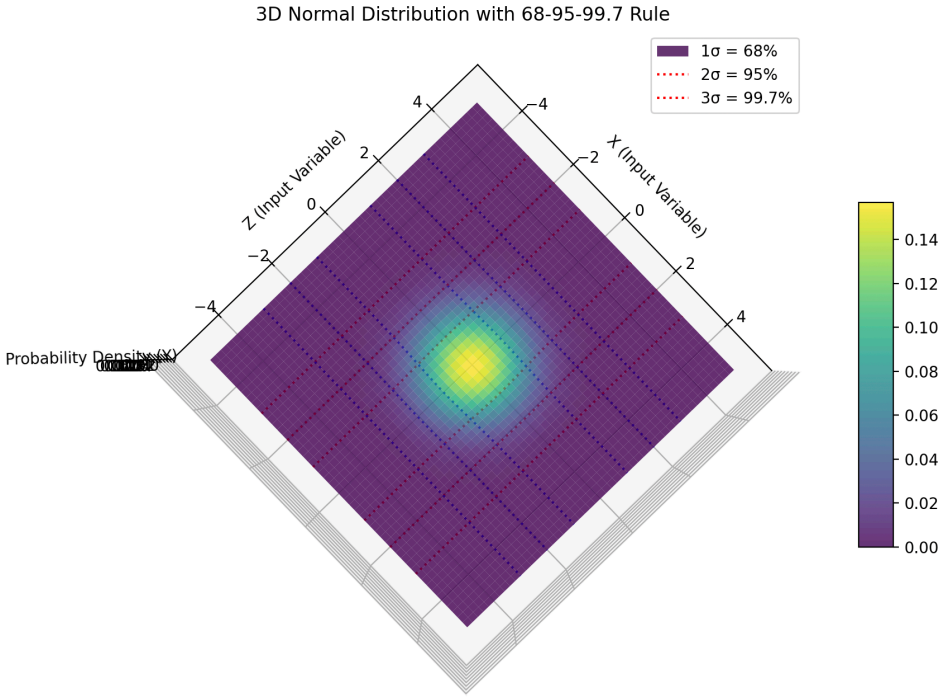
Appendix 9

Normaldistributioin.py output (no input required)

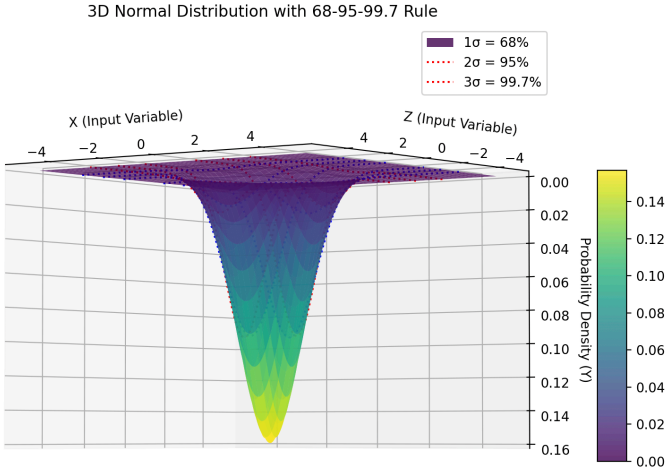
Angle 1



Angle 2

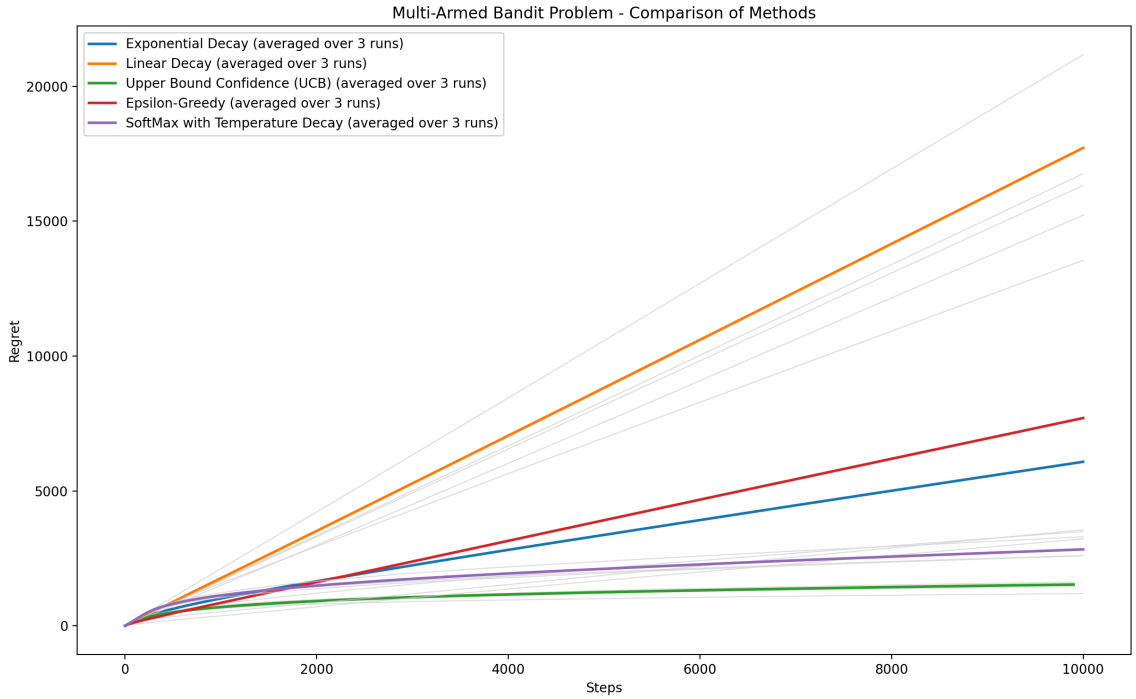


Angle 3



Appendix 10

Numerical regret



Appendix 11

Regretmulti.py script

```
import matplotlib.pyplot as plt
import numpy as np
import math
import random

def pad_rewards(rewards, target_length):
    return rewards + [float('nan')] * (target_length - len(rewards))

def exponential_decay(machines, spins, BanditMeans):
    epsilon_0 = 1.0
    decay_rate = 0.01
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        epsilon = epsilon_0 * math.exp(-decay_rate * i)
        if random.random() > epsilon:
            chosen_machine = np.argmax(rewards)
        else:
            chosen_machine = random.randint(0, len(rewards) - 1)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
```

```
        acr.append(cumulative_avg_reward)

    return acr

def linear_decay(machines, spins, BanditMeans):
    epsilon_0 = 1.0
    decay_rate = 0.01
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        epsilon = epsilon_0 * (decay_rate / (i + 1))
        if random.random() > epsilon:
            chosen_machine = np.argmax(rewards)
        else:
            chosen_machine = random.randint(0, len(rewards) - 1)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr

def ucb_method(machines, spins, BanditMeans):
    constant = math.sqrt(2 * math.log(spins) / machines)
    rewards = [0.0] * machines
    counts = [0] * machines
    acr = []

    for i in range(machines):
        reward = np.random.normal(BanditMeans[i], 1)
        rewards[i] += reward
        counts[i] += 1

    for j in range(machines, spins):
        upper_bound_all = []
        for i in range(machines):
            avg_reward = rewards[i] / counts[i]
             exploration_term = constant * math.sqrt((2 * math.log(j + 1)) /
counts[i])
            upper_bound = avg_reward + exploration_term
            upper_bound_all.append(upper_bound)
```

```
        selected_machine = np.argmax(upper_bound_all)
        reward = np.random.normal(BanditMeans[selected_machine], 1)
        rewards[selected_machine] += reward
        counts[selected_machine] += 1

        avg_reward = sum(rewards) / sum(counts)
        acr.append(avg_reward)

    return acr

def epsilon_greedy(machines, spins, BanditMeans, epsilon=10):
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        if i % epsilon != 0:
            chosen_machine = np.argmax(rewards)
        else:
            chosen_machine = random.randint(0, len(rewards) - 1)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr

def softmax_decay(machines, spins, BanditMeans):
    T0 = 1.0
    rewards = [0.0] * machines
    pulls = [0] * machines
    acr = []
    cumulative_reward = 0.0

    for i in range(spins):
        T = max(T0 * np.exp(-(i**2 / spins)), 1e-10)
        probs = []
        est_reward = []

        max_reward = max(rewards)
        for j in range(machines):
             est_reward.append(np.exp(((rewards[j] - max_reward) / (i + 1)) /
T))
        total_est_reward = sum(est_reward)
```

```
        probs = [er / total_est_reward for er in est_reward]

        chosen_machine = np.random.choice(machines, p=probs)

        reward = np.random.normal(BanditMeans[chosen_machine], 1)
        rewards[chosen_machine] += reward
        pulls[chosen_machine] += 1
        cumulative_reward += reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr

def hybrid_exponential_ucb(machines, spins, BanditMeans):
    epsilon_0 = 1.0  # Initial exploration rate
    decay_rate = 0.01  # Exponential decay rate
    rewards = [0.0] * machines
    counts = [0] * machines
    acr = []
    cumulative_reward = 0.0

    # Initialize each machine once (ensuring UCB starts with valid values)
    for i in range(machines):
        reward = np.random.normal(BanditMeans[i], 1)
        rewards[i] += reward
        counts[i] += 1
        cumulative_reward += reward

    # Main loop for action selection
    for i in range(machines, spins):
         epsilon = epsilon_0 * math.exp(-decay_rate * i)  # Exponential decay
exploration rate

        if random.random() < epsilon:
            # Exploration: Choose a random machine
            selected_machine = random.randint(0, machines - 1)
        else:
            # Exploitation: Use UCB without knowing total spins
            upper_bound_all = []
            for j in range(machines):
                avg_reward = rewards[j] / counts[j]
                   exploration_term = math.sqrt((2 * math.log(sum(counts))) /
counts[j])  # Uses past observations only
                upper_bound = avg_reward + exploration_term
                upper_bound_all.append(upper_bound)

            selected_machine = np.argmax(upper_bound_all)
```

```
        # Simulate reward
        reward = np.random.normal(BanditMeans[selected_machine], 1)
        rewards[selected_machine] += reward
        counts[selected_machine] += 1
        cumulative_reward += reward

        # Update cumulative average reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr

def main():
    machines = int(input("Enter the number of machines: "))
    spins = int(input("Enter the number of spins: "))
    repeats = int(input("Enter the number of repetitions for each method: "))

    BanditMeans = [np.random.normal(0, 1) for _ in range(machines)]

    methods = {
        '1': ("Exponential Decay", exponential_decay),
        '2': ("Linear Decay", linear_decay),
        '3': ("Upper Bound Confidence (UCB)", ucb_method),
        '4': ("Epsilon-Greedy", epsilon_greedy),
        '5': ("SoftMax with Temperature Decay", softmax_decay),
        '6': ("Hybrid Exponential-UCB", hybrid_exponential_ucb)
    }

    print("Select methods to run (separate by commas for multiple):")
    for key, (name, _) in methods.items():
        print(f"{key}: {name}")

    selected_methods = input("Your choice: ").split(',')

    plt.figure(figsize=(12, 8))
    plt.xlabel('Steps')
    plt.ylabel('Regret')

    for method_key in selected_methods:
        if method_key.strip() in methods:
            method_name, method_func = methods[method_key.strip()]
            all_runs = []
            cumulative_rewards = np.zeros(spins)

            for _ in range(repeats):
                rewards = method_func(machines, spins, BanditMeans)
                rewards = pad_rewards(rewards, spins)
                all_runs.append(rewards)
```

```
                cumulative_rewards += np.nan_to_num(rewards)
                regret = []
                regret.append(rewards[0])
                for i in range(1, spins):
                    regret.append((max(BanditMeans) - rewards[i]) + regret[i -
1])
                plt.plot(regret, color='lightgray', linewidth=0.8, alpha=0.7)

            cumulative_rewards /= repeats

           cumulative_rewards = np.where(cumulative_rewards == 0, float('nan'),
cumulative_rewards)
            avg_regret = []
            avg_regret.append(cumulative_rewards[0])
            for i in range(1, spins):
                avg_regret.append((max(BanditMeans) - cumulative_rewards[i]) +
avg_regret[i - 1])
           plt.plot(avg_regret, label=f"{method_name} (averaged over {repeats}
runs)", linewidth=2)

    plt.title("Multi-Armed Bandit Problem - Comparison of Methods")
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

Appendix 12

Linear Decay Script

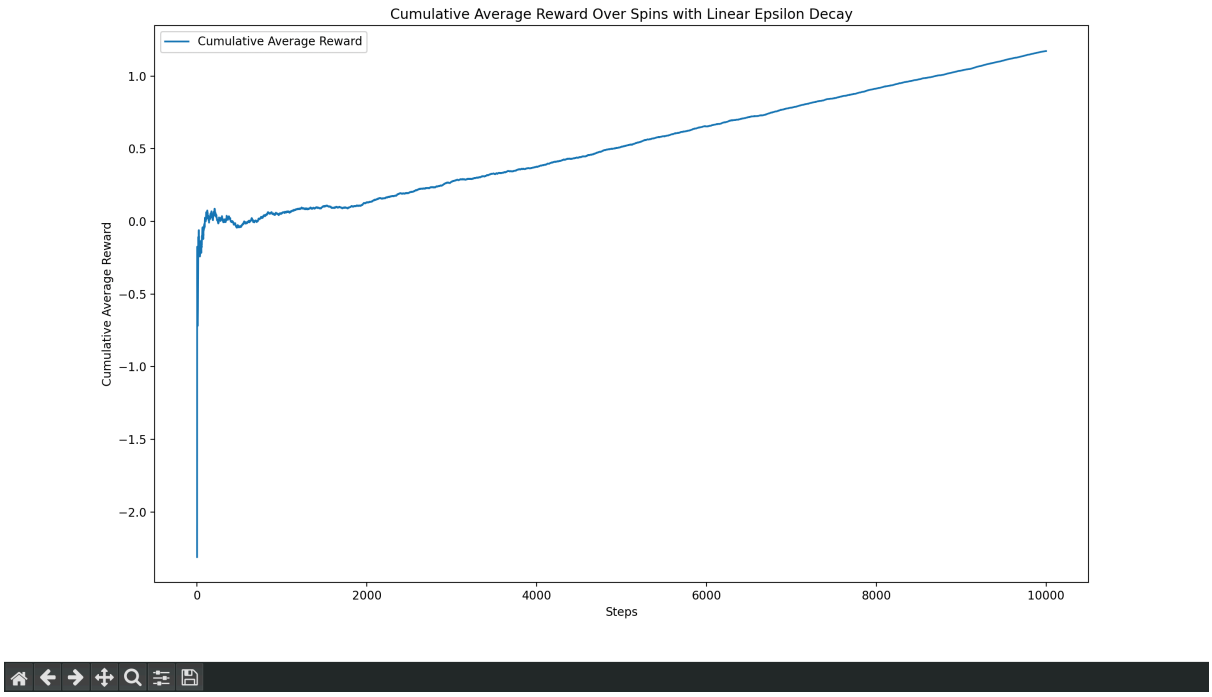
```
import matplotlib.pyplot as plt
import numpy as np
import random


#extremely high dependance on the initial spins for effecitiveness of algoritm
plt.figure(figsize=(20, 16))
plt.xlabel('Steps')
plt.ylabel('Cumulative Average Reward')
```

```
variance = 1
machines = int(input('Machines - '))
spins = int(input('Spins - '))

#this is the
epsilon_0 = 1.0
epsilon_min = 0.01
decay_rate = (epsilon_0 - epsilon_min) / spins

BanditMeans = [np.random.normal(0, 1) for _ in range(machines)]
rewards = [0.0] * machines
pulls = [0] * machines
acr = []

cumulative_reward = 0.0
for i in range(spins):

    epsilon = max(epsilon_min, epsilon_0 - decay_rate * i)


    if random.random() > epsilon:

        chosen_machine = np.argmax(rewards)
    else:

        chosen_machine = random.randint(0, len(rewards) - 1)


    reward = np.random.normal(BanditMeans[chosen_machine], variance)
    rewards[chosen_machine] += reward
    pulls[chosen_machine] += 1


    cumulative_reward += reward
    cumulative_avg_reward = cumulative_reward / (i + 1)


    acr.append(cumulative_avg_reward)


plt.plot(acr, label='Cumulative Average Reward')
plt.title('Cumulative Average Reward Over Spins with Linear Epsilon Decay')
plt.legend()
plt.show()
```

Appendix 13

Linear Decay (input and output)

```
Machines - 100
Spins - 10000
```



Appendix 14

Exponential Decay Script

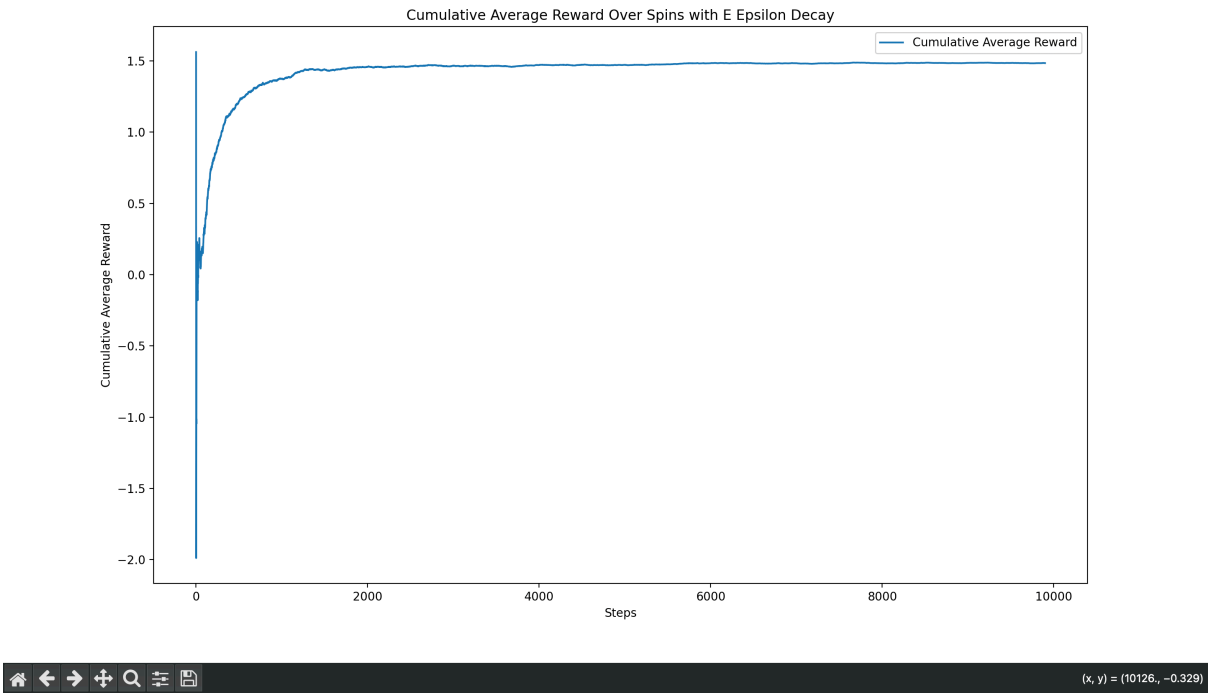
```
import matplotlib.pyplot as plt
import numpy as np
import random
import math

#extremely high dependance on the initial spins for effecitiveness of algoritm
plt.figure(figsize=(20, 16))
plt.xlabel('Steps')
plt.ylabel('Cumulative Average Reward')

variance = 1
machines = int(input('Machines - '))
spins = int(input('Spins - '))


epsilon_0 = 1.0
decay_rate = 0.01


BanditMeans = [np.random.normal(0, 1) for _ in range(machines)]
rewards = [0.0] * machines
pulls = [0] * machines
acr = []
practice = machines
cumulative_reward = 0.0



for i in range(spins-practice):

    epsilon = epsilon_0 * math.exp(-decay_rate * i)


    if random.random() > epsilon:
```

```
        chosen_machine = np.argmax(rewards)
    else:

        chosen_machine = random.randint(0, len(rewards) - 1)


    reward = np.random.normal(BanditMeans[chosen_machine], variance)
    rewards[chosen_machine] += reward
    pulls[chosen_machine] += 1


    cumulative_reward += reward
    cumulative_avg_reward = cumulative_reward / (i + 1)


    acr.append(cumulative_avg_reward)


plt.plot(acr, label='Cumulative Average Reward')
plt.title('Cumulative Average Reward Over Spins with E Epsilon Decay')
plt.legend()
plt.show()
```

Appendix 15

Exponential Decay (Input and Output)

```
Machines - 100
Spins - 10000
```

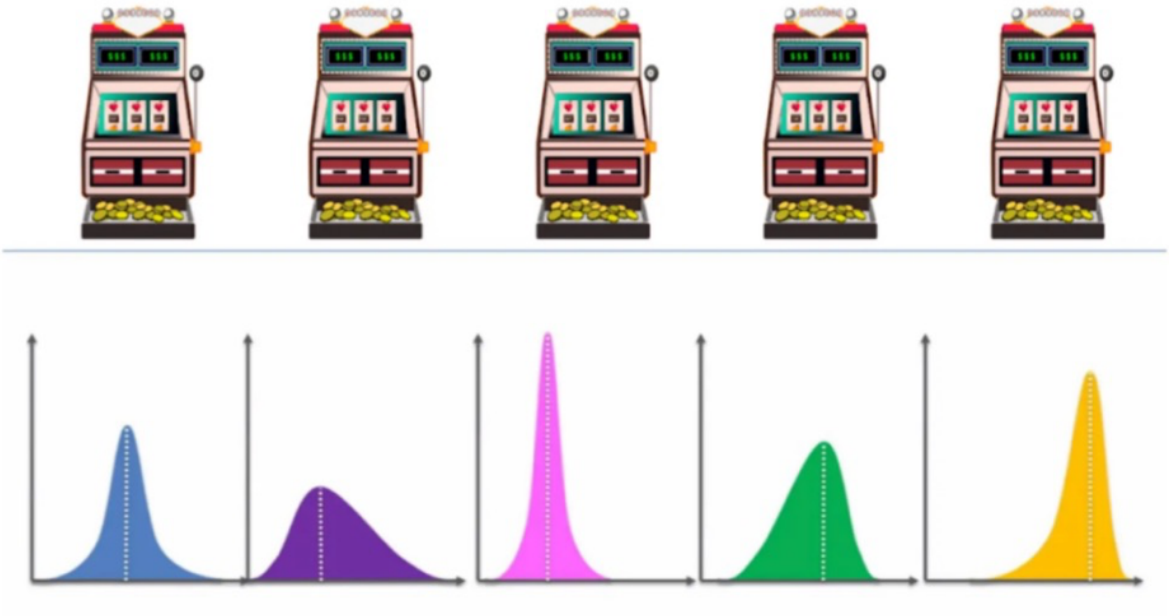


Appendix 16

MAB visual representation

Appendix 17

MAB visual representation



Appendix 18

Script for trading algorithm (UCB method and Q learning)

```
import numpy as np
import yfinance as yf
import time
from datetime import datetime
import matplotlib.pyplot as plt

# Q-Learning Trader Class
class QLearningTrader:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
```

```
        self.q_table = np.zeros((state_size, action_size))
        self.learning_rate = 0.1
        self.discount_factor = 0.95
        self.exploration_rate = 1.0
        self.exploration_decay = 0.995

    def choose_action(self, state):
        if np.random.rand() <= self.exploration_rate:
            return np.random.randint(self.action_size)  # Explore
        return np.argmax(self.q_table[state])  # Exploit

    def update_q_table(self, state, action, reward, next_state):
        best_next_action = np.argmax(self.q_table[next_state])
        target        =        reward        +        self.discount_factor        *
self.q_table[next_state][best_next_action]
        self.q_table[state][action]    +=    self.learning_rate    *    (target    -
self.q_table[state][action])

    def decay_exploration(self):
        self.exploration_rate *= self.exploration_decay

# Fetch intraday stock data (1-minute intervals)
def fetch_intraday_data(ticker):
    stock = yf.Ticker(ticker)
    data = stock.history(interval="1m", period="1d")  # Get 1-minute data for 1 day
    if data.empty:
        raise ValueError("Invalid ticker. Please enter a valid stock symbol.")
    return data['Close'].values  # Return the closing prices at each minute

# Simulation function for training the Q-Learning model
def simulate_intraday_trading(data, trader):
    total_profit = 0
    state = 0

    for t in range(len(data) - 1):
        action = trader.choose_action(state)
        next_state = state + 1

        if action == 0:  # Buy
            reward = data[t + 1] - data[t]  # Profit if the price goes up
        elif action == 1:  # Sell
            reward = data[t] - data[t + 1]  # Profit if the price goes down
        else:  # Hold
            reward = 0

        trader.update_q_table(state, action, reward, next_state)
        total_profit += reward
        state = next_state

        if state >= trader.state_size - 1:
            break
```

```
    return total_profit

# Live trading function with 20-second interval decisions, partial buy/sell, and
enhanced algorithm
def live_intraday_trading(ticker, trader, initial_cash=1000, duration_minutes=10):
    state = 0  # Start at the first state
    cash = initial_cash  # Starting cash
    shares = 0  # Starting with no Tesla shares
    last_price = None  # Track the last price for buy/sell decisions
    initial_time = time.time()  # Start time
    trade_history = []  # Store history of actions for plotting
    cumulative_profit = 0  # Track profit
    assets_history = []  # Track assets over time (cash + shares value)

    while time.time() - initial_time < duration_minutes * 60:    # Run for
`duration_minutes` minutes
        # Fetch live data (use closing prices in intervals)
        live_data = fetch_intraday_data(ticker)
        price = live_data[-1]  # Latest price (1-minute interval data)

        if last_price is not None:
            action = trader.choose_action(state)

            # Decide how much to buy/sell (50% of available cash/shares)
            amount_to_trade = 0.5  # Trade 50% of available cash/shares

            # Buy a fraction of available cash
            if action == 0 and cash >= price:
                shares_to_buy = int((cash * amount_to_trade) // price)
                cost = shares_to_buy * price
                if shares_to_buy > 0:
                    shares += shares_to_buy
                    cash -= cost
                    print(f"Buy {shares_to_buy} {ticker} at ${price:.2f} (Timestamp:
{datetime.now()}), Shares: {shares}, Cash: {cash:.2f}")
                    trade_history.append((state, price, "buy"))

            # Sell a fraction of owned shares
            elif action == 1 and shares > 0:
                shares_to_sell = int(shares * amount_to_trade)
                if shares_to_sell > 0:
                    cash += shares_to_sell * price
                    shares -= shares_to_sell
                    print(f"Sell {shares_to_sell} {ticker} at ${price:.2f}
(Timestamp: {datetime.now()}), Shares: {shares}, Cash: {cash:.2f}")
                    trade_history.append((state, price, "sell"))

            # Hold
            else:
```

```
                print(f"Hold {ticker} at ${price:.2f} (Timestamp: {datetime.now()}),
Shares: {shares}, Cash: {cash:.2f}")
                trade_history.append((state, price, "hold"))

            # Calculate reward based on profit/loss
            profit = (price - last_price) * shares
            cumulative_profit += profit
            reward = profit if profit > 0 else -1 * abs(price - last_price)  #
Penalize loss more severely
            trader.update_q_table(state, action, reward, state + 1)

        # Track total assets (cash + value of shares)
        total_assets = cash + shares * price
        assets_history.append((state, total_assets))

        # Update state and last price
        state += 1
        last_price = price

        # Wait for 20 seconds before making the next decision
        time.sleep(20)

    # Display cumulative profit at the end
    print(f"Total Profit: ${cumulative_profit:.2f}")
    return trade_history, cumulative_profit, assets_history

# Function to plot buy/sell/hold actions over time
def plot_trades(trade_history, assets_history):
    states, prices, actions = zip(*trade_history)
    asset_states, asset_values = zip(*assets_history)

    buy_states = [s for s, a in zip(states, actions) if a == "buy"]
    buy_prices = [p for p, a in zip(prices, actions) if a == "buy"]

    sell_states = [s for s, a in zip(states, actions) if a == "sell"]
    sell_prices = [p for p, a in zip(prices, actions) if a == "sell"]

    hold_states = [s for s, a in zip(states, actions) if a == "hold"]
    hold_prices = [p for p, a in zip(prices, actions) if a == "hold"]

    plt.figure(figsize=(10, 6))
    plt.plot(states, prices, label="Price", color="gray")
    plt.scatter(buy_states,  buy_prices,  label="Buy",  color="green",  marker="^",
alpha=1)
    plt.scatter(sell_states,  sell_prices,  label="Sell",  color="red",  marker="v",
alpha=1)
    plt.scatter(hold_states,  hold_prices,  label="Hold",  color="blue",  marker="o",
alpha=0.5)

    # Plot the total assets over time
```

```
    plt.plot(asset_states, asset_values, label="Assets (Cash + Shares Value)",
color="orange", linestyle="--")

    plt.title("Buy, Sell, Hold Actions and Assets Over Time")
    plt.xlabel("Time Steps")
    plt.ylabel("Price / Assets")
    plt.legend()
    plt.show()

# Main function to train and run the short-term trading
if __name__ == "__main__":
    # Ask for stock ticker and validate it
    while True:
        ticker = input("Enter the stock ticker to trade (e.g., TSLA): ").upper()
        try:
            stock_data = fetch_intraday_data(ticker)
            break
        except ValueError as e:
            print(e)

    state_size = len(stock_data) - 1  # Number of states (time steps)
    action_size = 3  # Actions: Buy, Sell, Hold

    # Initialize and train trader
    trader = QLearningTrader(state_size, action_size)

    # Simulate trading for 1000 episodes to train the model
    num_episodes = 1000
    for episode in range(num_episodes):
        simulate_intraday_trading(stock_data, trader)
        trader.decay_exploration()

    print("Training completed.")

    # Ask the user for how long they want to run the live trading (in minutes)
    duration_minutes = float(input("Enter the duration to run the live trading (in
minutes): "))

    # Start live trading with 20-second intervals and initial cash of $1000
    trade_history, total_profit, assets_history = live_intraday_trading(ticker,
trader, initial_cash=1000, duration_minutes=duration_minutes)

    # Plot the trading actions, prices, and assets
    plot_trades(trade_history, assets_history)

    # Print final profit
    print(f"Final Total Profit: ${total_profit:.2f}")
```

Appendix 19

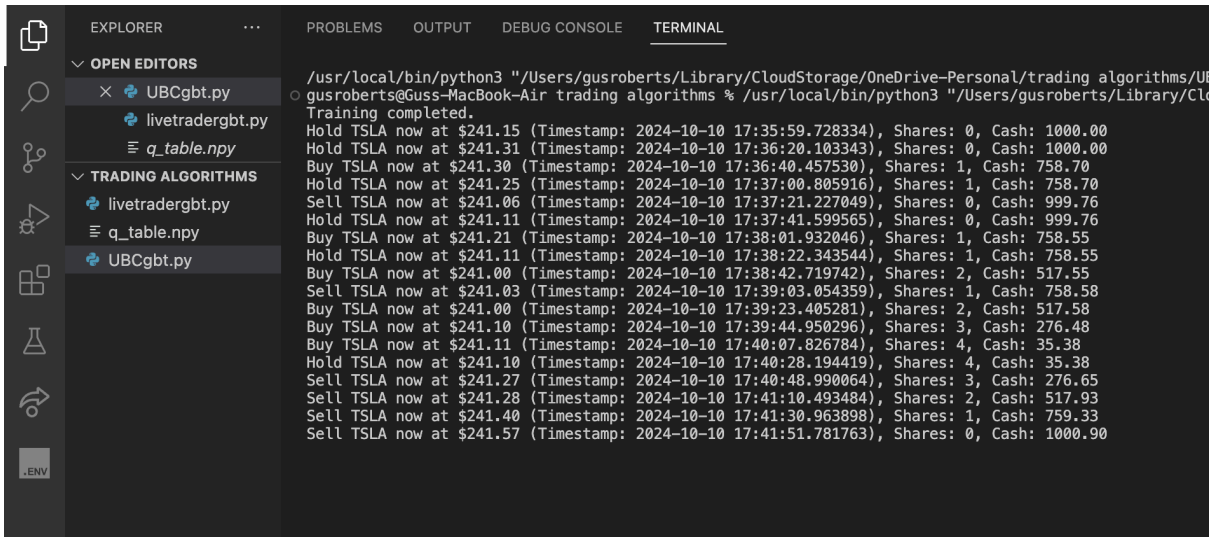
Sample of Q table for UCB HFT algorithm


```
NUMPYv{'descr': '<f8', 'fortran_order': False, 'shape': (1257, 3), }
1          m                        @          (                                  @
"\y@     @ k d  @6 ئ   @   ! @`/@ @ G\ ւ@ U &  @z – )@3<  "ς@   ת @Eq  ∗ @  /   @  7
@ے  C5 @ ỲZ   @9< ´ @Tn0   @   Ss] @ Ï д @ C3ڑ  @XX ( @ŭg5ь@vn O' @ /F ∗ @gn @ ^v
@7  2 B @
  F  @LL  @   a
 @   c{2 @5      #  @ :    @ T _c @'1, @ 6z ³ @ 5    @oO ~  @\   A□@ g" P @ u[ `  @
,   s~      @f   R8G@O   5@WT           @           f   @&   s   H   @   |   uj   @
 @+@  _
 @ Y(Vqa @  0 D @kHkDˈ@    u  @   – @  EuNy @`   7  @  y +Yp@$>W#E @   ]ŏ @  m  □@ 8
\5& @  :–◌@VA+  @ b   |@  r2 @w'8T– @idˊ {@}^ D @  c
 @ᵘ ?DH□@f p$]@2    oz@. ]bg @gu 4K□@ `[I<z@ 0    @m3c  @\ A  < @Phx @F◌5ς@ r!  ~@ ւ
, @  X Y' @N N .  @oőI  @T[6  D}@ u      @d2% FO|@  nh @∗ b4D  @>K   l~@W L  ~@_ W
8{@T   / @ 4 ! _ @; (>; @  ;p ɟ@ Z –  @ʊ ^3 @j[  a= @□ c  – @  j  k @>λ T◌@i 4  v @
□<     ]     @T          [o     @G               Y     @!W@          E     F     @
 U   @ Ä3B% @YHh   @ ت z  @@\CZNϡ@0J
&  @? p  + @5 + }@z!  @    T @<   f^ @∗M @      p @m 3"^y @X    5 @ k  |< @Z ∗eS @a
"f   @ 콧 l @ \   – @  k  ` @%  [z @  g – @   u{ @    _@^K– – @ u`' @x    @s Ó @9 [@
) @ f#!ے@ □ @ rx  . @f | ; @h 2  @ □ {r6@ڃ  đ@V◌  @ xL 3 @`6    @$ ض U @ŋey[  @=G %
@~   @ a N_  @.Ţ    @ [ O  @tê   @
+   @

 @~"K  ~ @E  v Q @|m∗# @i    @z ]H   @ Wk @D7     @i E Ɛ@  vb – @ ő V @ A   @   ( @
@:                                                                          wX
 @2 MX @   uy  @  A fz @ ]ᑕG @ "@8_  @     @ + Ūʌ@1C { E @O∗5ᴛ@ ɟ õ@ᴐc□& @^if3 O @<
â) @_ 1
  @
   @ I  v@ ϕb @ e# 1
}@ o  p @  ^  ڄ@R[  2  @    @9R pW @(ˊ iH @  u ; {@ .   @J Ad ւ@x ?  @   q/  @  fPF□@o
{z@(
G   @r  □   @  '   8  @  xWk□@  v     j  @  nI  □@M     6   @  s  e,O}@  Ш  ~}@  }i@L
 @  r [ ~@z'VeY  @a   $) @i c  □@&  m]h @p~    @   + @S# □ @V P  ◦@\ :□ @>  mJ` @.Y
<                  @V      Y                  @                          +□      @
8□□@                        ,     5     @                                □@     ZW
G @    □@  ]x} @ –  zr @b~A   @ vw @    t □@D?   @ c} P  @  m– @  F^  @e  ∗ ^ @ Z
@              U          @          1[     m     @;                          @4^K
. @ė d + @K  w @  @  ˙@ ː y @G_= h  @n] 8d @LQ  I @/ ʾɟ@  }x`1 @ i   @   A j @  S   @
0J    @r  ~         @    |  Ⱶ@    6    ц@~f  @  MU  U    @<e–1      @  l      Ỏ@
]8_"p @Ǹ   ŏ@ic     @= q p  @ 5 N @ " ◌̄w @   ?  @ {Jx@W–M H @ + =□ @ ŀ t  @{
  ´H@s       y[j   @>   JB꜀   @   4         @   <          @E·tdÓ@   Wh   @Y)$
`      @M      @                  @           Z                  @0           }
  @ 6: c  @I "    @? nsy  @ Rg    @ j  @  3  @. =   @□Xڄ@m @]I i  @ ւ9 V @d  J成@h
' O @uK @fo:"耘@FM   K @ 2$81  @
   @ :K_ @   A e @ {   @ t !  @ `    @ –@ H4 @ʾTPʒ @ J   d @  TgR̂@ 3 8p  @ 8   @ L:d
= @" M  @K y  @    V  @  e   @   JM @□8   ˣ@. + e  @갅    @□ ɒ)  @ [MB@ Yz~ @Q  毀@
u' @gs1[js @ %) S @ V     @@ )6Y @ ; #G @  7
 @     @{  <   @3 k   ~@ n     @  z @;L@  – @Z   + @ 4 s k @  ]k @   µ @       i @
$  z @  8ʏ  @"0□  @ ∢+ @&x.T @    # L   @ µ–_@ u R   @u  oϑ@ >Â@  7   @My^ Me @, _
ڄ@     @N F– @î   . @ j7  J@t ,v      @ Ǣ0 0 @JL ng  @=>ے@ ,  p  @H  昜@ # 3TX @4
```

/kꜜ@z_ Ì @ 3 È @ A 8 o @ 稔@9 kt @ QU @ ` W/ @ ِ & @iD I @_u w Ꮆ@ Qe @ u ?
@ᵖ . @ wy @& H0 @f @ u Ꮆ@W u ∗D @ I Ō@E p @\ a @ ᑭ @ [V8
Œ@w U @ w t @.9 @ F & N @{
wVmS @i Rf>(@ | 6N @c}m M @ дк¶ @H[[@ p8
@Q'p; ? @嚏K(@ t%蝐@qm /− @TLx+u @ O' @ Y H @ @e 8o @ň X @ @w @
Ub @ ,ǐ@R kR @:c n]f @ ^@ E꞉@X @) }< @I $ ⊥@ ㄒ@d |y@?b# K @ @`g;
@k @ J jj @ 7 . @ ى@ W j @∗ 1ъ @∗G # @n } @ U⅍4 @ s
O @ ^e−B @ \∗ @− 2 − @ E @syKyƖ@ A;VpQ @ 23 @ T N+K @ E怨@ R @ wx @ڀ
5 @ 9 @8;5 θ@1 m h @g:7 z @ ţ͞ū@7gL H @x d O @ | M @ 5 @ y " @ E
@|'wd
@ S Q. @|S i @4 /L @Z¬ h @ y ڑ
@a T @ b ᴘ @/ @ @ J 4 @Q0Mdc @E ے @ S:=ᦒ@ S uR @ ' P @ 0 D @
hH Ώ @ 90 % @V' − @ Ґ@ cg @ r + 1 @ c /p @e = @ CM"\F@V}1u @‿] @ 2 (˚@ _
@$ v> @ D lL @□O7n @ ꞁ S @ o @; @ 0 @ l
@ ڜ) @ V8
% @
"g @Ÿ桦7 @& @èO x @8 b @`∗ bl @|} # @O T @ G@g&=— @ □ @ < T @o ь@)
}(@>s è@
d @ ^ jv @T [9 , @ Vm ᒥ@dS∗a− @| l @ Cb @ ɴ A` @J L k @RLk= @ %[Z@
p @G 9Yй @ ∗3a J @ ?Z @ 1 @y|e V @ iû@ (B @. 9| @ J ∗Q @ □ @ k
@ ? J˚@^ # @ <> @ w @ N L @ B3 G @0 4@ ڸ!□ W @ L " @ B, @E `Sh @
w <a @ ~ o ä@^ r @ I5{
@−YKB @kV C t @ @ 7 "˙@ rZ @ 1qZp`@ N @ q @ 0R 0 @ὁ hK @ 5 , @ ~] @
−n @ ?𝌆@9HAmk! @ ~%й@ LC + @ %B
w @ i) − @WSxFY> @∗ , @oB , @
b @ Eux @.NE7 @ oT} ⸝@aM? t @ 6 @) @ y ∗f @B t r @, ;Or @ t @ $ N
@IR w뵑@q=hD @ zM8@ₔ `; @ !W綩@ 7 @+: V @ 7]l @ n @ ◌̈@ @ s @:yV
@ |@ { @ □ @TS K4 @Q B|A @ ∗ @젖 ,− @ 9@ + @ᵢ @ @ @R @D7
@ cK @_∗ o 8 @ □ @f T @ j P[h @ ˟ @P3 s @\−p t! @ B%Uz~ @+ #{□@1ὁ
@ Ǻ x @! @ 8 CH @ @□ ج@.Q r @ v ق@ lL@ Z3 @□ 7 ∗□@( )g = @ n? @ X+
@P"& @B(y) @? Q#H @ ᴛw @p& g ř@u u e @@l C P| @ ,]af
@tR G @P fq @G □ J @E ;{@6 K;@ K @ l @|y @ 7@ l #@B□ @ \ @?;
:@Xn − @ \fsz @ F`m @ X @ I @
@ "]e5 @ z @p }_< @ ڑ} k @ᐟ @Eᵛ %n @Dr @ J @ d e @ 2n□ @ j @> B9f
@ j a; @
LI @ x x ∗ @X h @ Q B @=Z
@□@8s C @ h}fh @ $ i @ ﻩ z: @<
yXϑ@ g?3 @ +\□@JT3W k @a2 |% @T\ @恢ˈ@QBH@K: @ □FF @# u Q @ { @7 qY{. @s
@d 1ѽӊ@a cL @0Z %ج @R ;W A @ W ` @' 饈@ > ӊ@ `c4 @5A`. @W = @ i @ @h
@ 8\ ِ@T z @ R : Ē@ 5IIP
@J
K2= @3 S @□f@G@ (@ j}@ [u} @ ɣ − @ 8Dr R @ j_ ❋@ }^ u @jK
@ w N @ $O N @ r% J @ y = @ #J⌟' @ @ ,J @ @8 ; T @ bB @1 − @= Qᴛ@ ~
@c`Xf @ wA; @ { 'Y @&5 @ d @ ?V> @ " j @ H | ´@ @p, @ P @6A s @k
d∗$ @6j D @ Fi G @ m wB @q! 3 @I
@ $}L @l1 !,(@ 8\ tK @ :
@ 5= @ c<p? @ \ □ @ k @ L @rb @ M @ &o3 @ b @ ! = @ q ъ@_s J @ B
$ @rK?uv(@l [ш {@ % @1@ ᵛq @ V @ @ @`y ف@>[f$ @ ∗2 镄@ JS − @ ! s @
R @ ∗ @w V @ l−~ @ Q iW @i ˇ@Y j Y‿@]
> @ ∗~{Nu @I @erI□ʅ@C' % @ Bc @ Fcfi @ Ί @ pwj @i ˆ@ aU @ E 3] @ ɟX.| @
J @ M @)g]o @Z$ ɣz @ʜ ,+O @ { q @ c|8ō@| l @ . @ ?[x @I 5Ru‿@ 1 XĄ@
p F @ ꞉ @ ' @ Oᵩ @ʜQ . @}ὁ @ IE @b Ngق @wd @Bꜝ̣Rp @ F O @ VJY □@ t 7
ʐ@ b Y @X>{ } @ jV @ h @ Ĉ@ " @+r i− @ + /X @. # @i JV @N 6d @< □
@ . + @ D§9 @
@ _ @ dB & @x# j @9S @ A/ @w L @J ᕒ @\} } ˈ@ d o @ r봌 @GUV ъ@ k
Nj@}&@
Q @^ J u @F x? @sl ʑ@ 7N P @\ ! W @ˌy Ң@
Ju @V @ 4 @<!
@t H n @[− @ 2 y @

Appendix 20

UBC HTF output



Appendix 21

ε linear decay formula

Mathematical Notation:

Let $r \sim U(0, 1)$ (a random value from a uniform distribution between 0 and 1).

$$\text{If } r > \epsilon, \text{ then}$$

$$\text{chosen\_machine} = \arg\max_j(R_j) \quad (\text{where } R_j \text{ represents the reward for machine } j).$$

$$\text{Otherwise,}$$

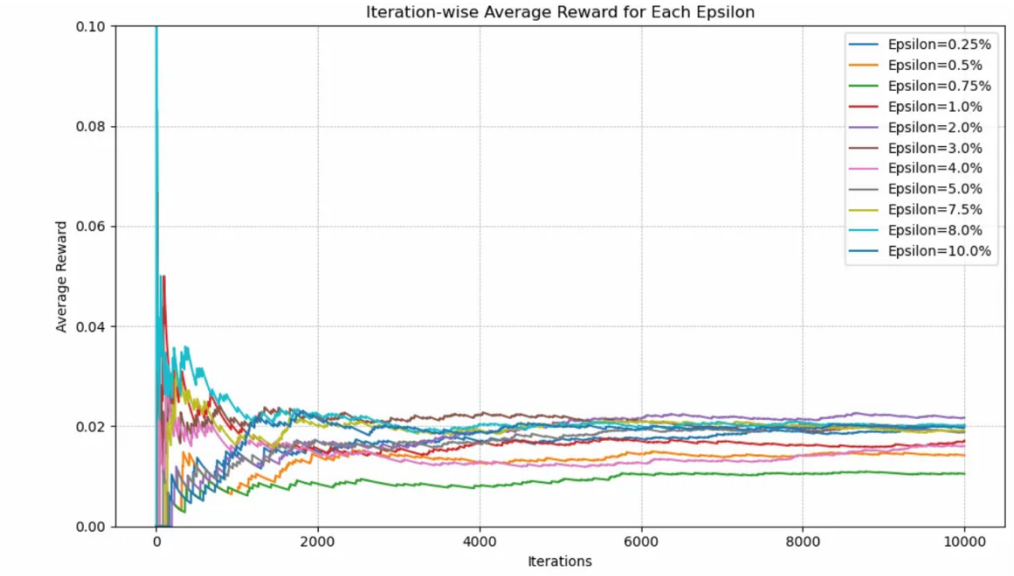
$$\text{chosen\_machine} \sim \text{DiscreteUniform}(0, n-1) \quad (\text{randomly choose an index from 0 to } n-1).$$

Explanation:

1. $r \sim U(0, 1)$: Represents the random number generation from a uniform distribution.
2. $r > \epsilon$: Corresponds to the exploration vs. exploitation decision.
3. $\arg\max_j(R_j)$: Select the index j with the maximum reward R_j (exploitation step).
4. $\text{DiscreteUniform}(0, n-1)$: Choose a random machine index uniformly from 0 to $n-1$ (exploration step).

Appendix 22

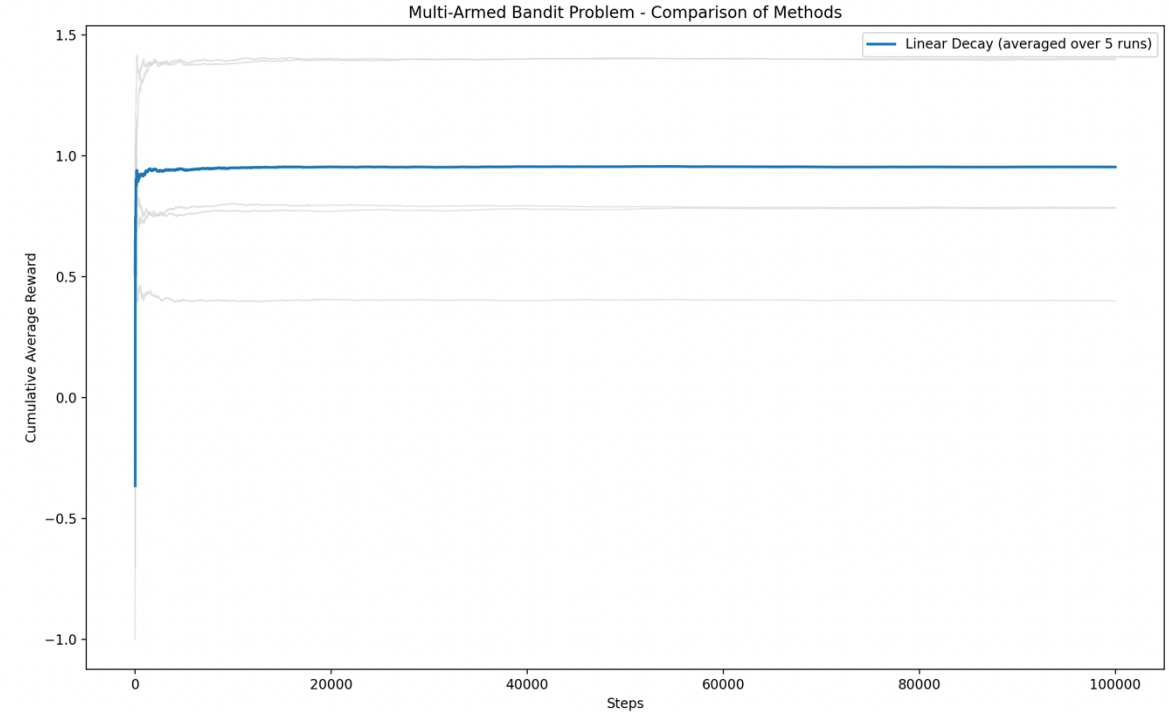
Optimal $\boldsymbol{\varepsilon}$ value for simple epsilon greedy



From https://medium.com/@gridflowai/part-2-in-depth-exploration-on-epsilon-greedy-algorithm-2b19e59bbe22

Appendix 23

ε (Linear decay) – 5 repeats



Appendix 24

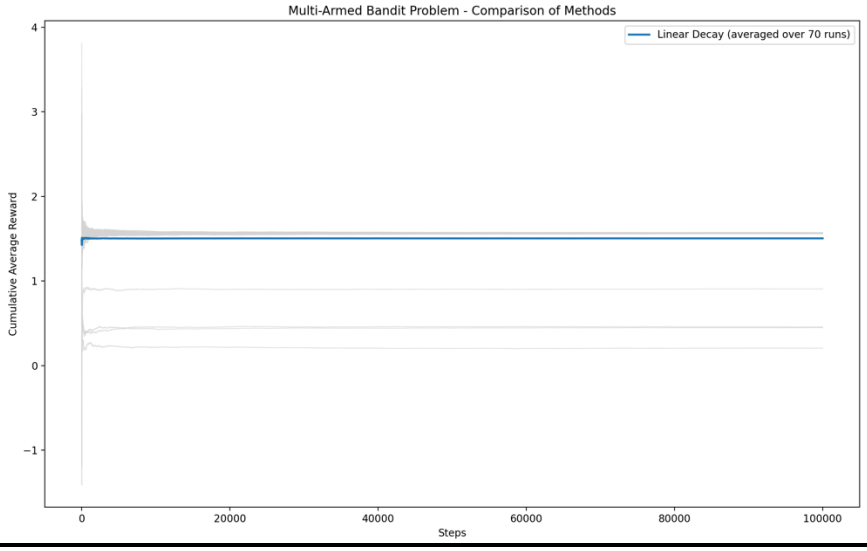
ε (Linear Decay) Script

```
import matplotlib.pyplot as plt
import numpy as np
import random


#extremely high dependance on the initial spins for effecitiveness of algoritm
plt.figure(figsize=(20, 16))
plt.xlabel('Steps')
plt.ylabel('Cumulative Average Reward')

variance = 1
machines = int(input('Machines - '))
```

```
spins = int(input('Spins - '))


epsilon_0 = 1.0
epsilon_min = 0.01
decay_rate = (epsilon_0 - epsilon_min) / spins

BanditMeans = [np.random.normal(0, 1) for _ in range(machines)]
rewards = [0.0] * machines
pulls = [0] * machines
acr = []

cumulative_reward = 0.0
for i in range(spins):

    epsilon = max(epsilon_min, epsilon_0 - decay_rate * i)


    if random.random() > epsilon:

        chosen_machine = np.argmax(rewards)
    else:

        chosen_machine = random.randint(0, len(rewards) - 1)


    reward = np.random.normal(BanditMeans[chosen_machine], variance)
    rewards[chosen_machine] += reward
    pulls[chosen_machine] += 1


    cumulative_reward += reward
    cumulative_avg_reward = cumulative_reward / (i + 1)


    acr.append(cumulative_avg_reward)


plt.plot(acr, label='Cumulative Average Reward')
plt.title('Cumulative Average Reward Over Spins with Linear Epsilon Decay')
plt.legend()
plt.show()
```

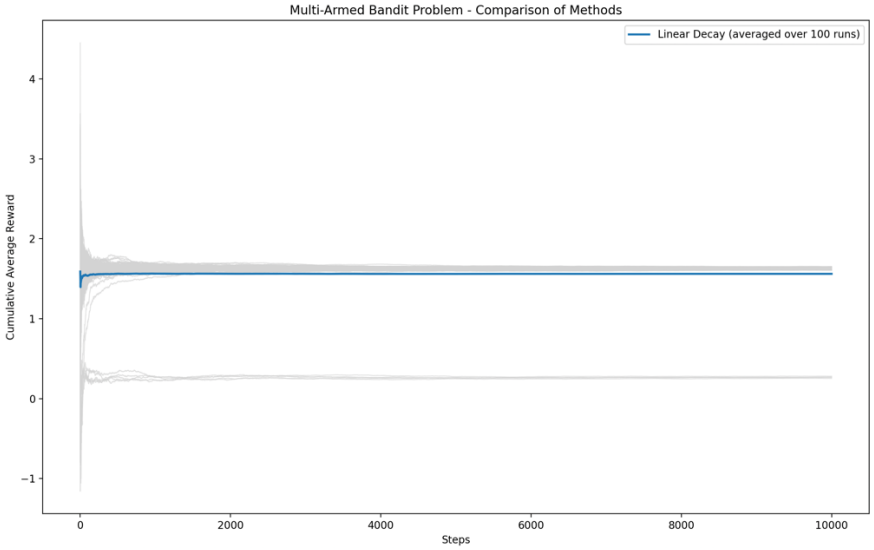
Appendix 25

ε (Linear Decay) – 70 repeats



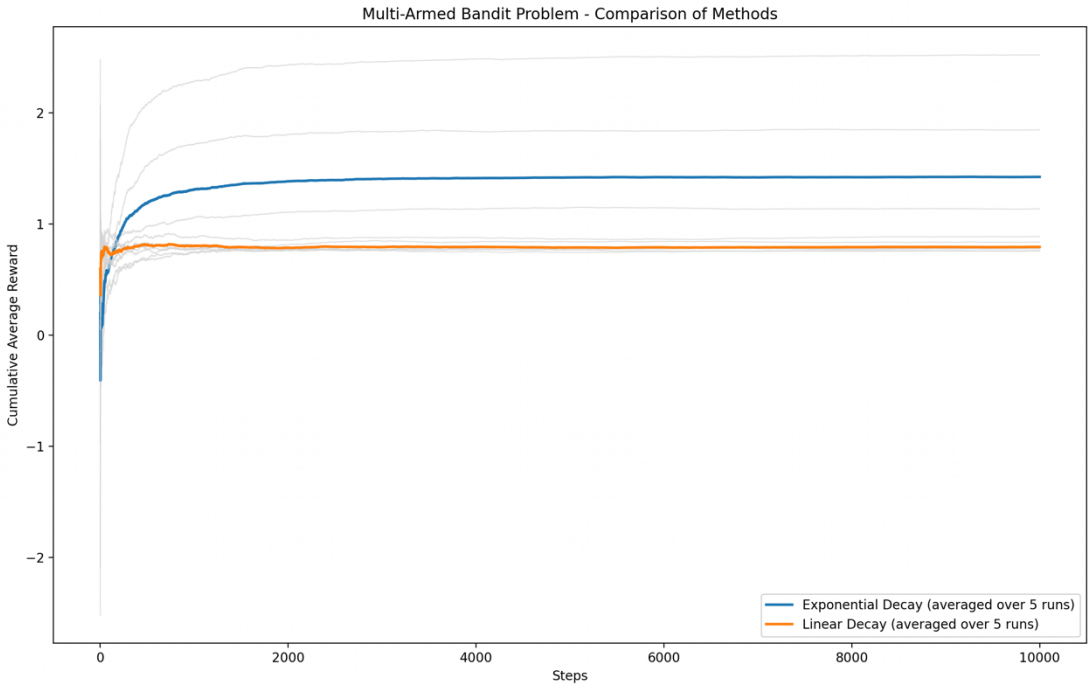
Appendix 26

ε (Linear Decay) – 100 repeats



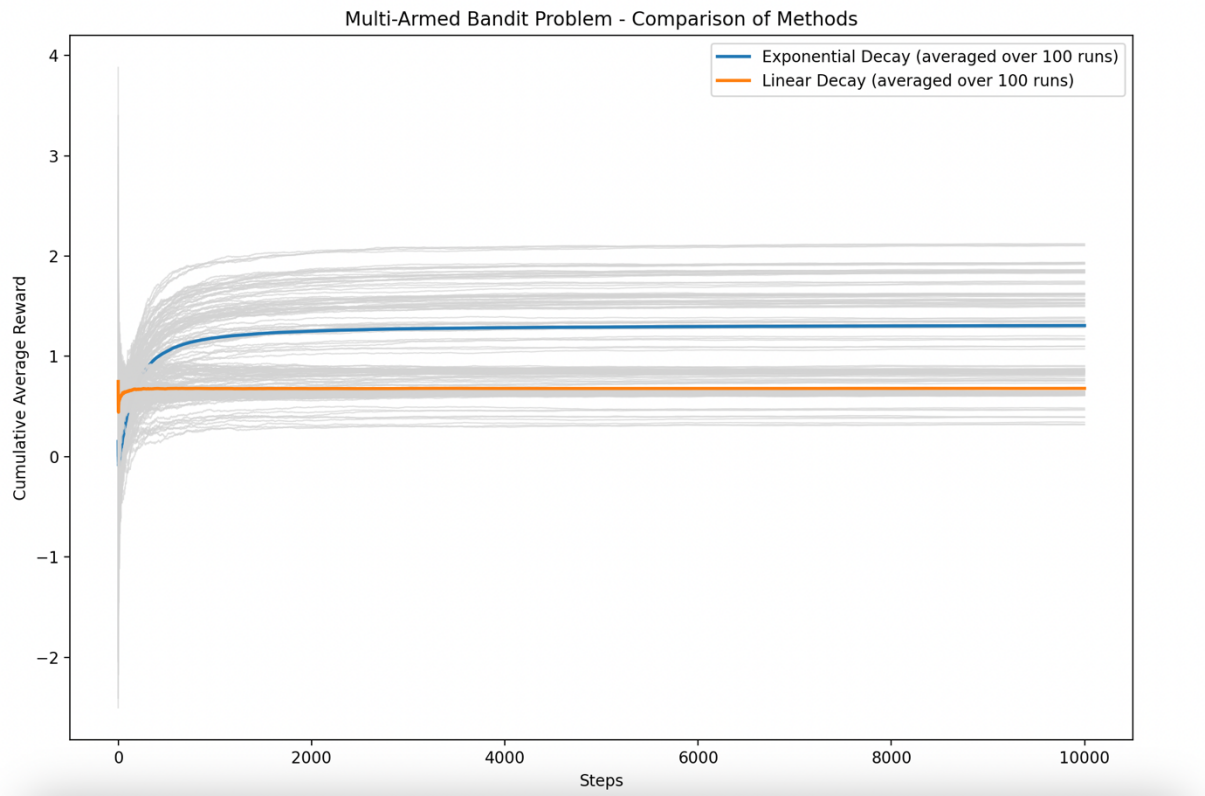
Appendix 27

Linear vs Exponential Decay output (structure comparison)



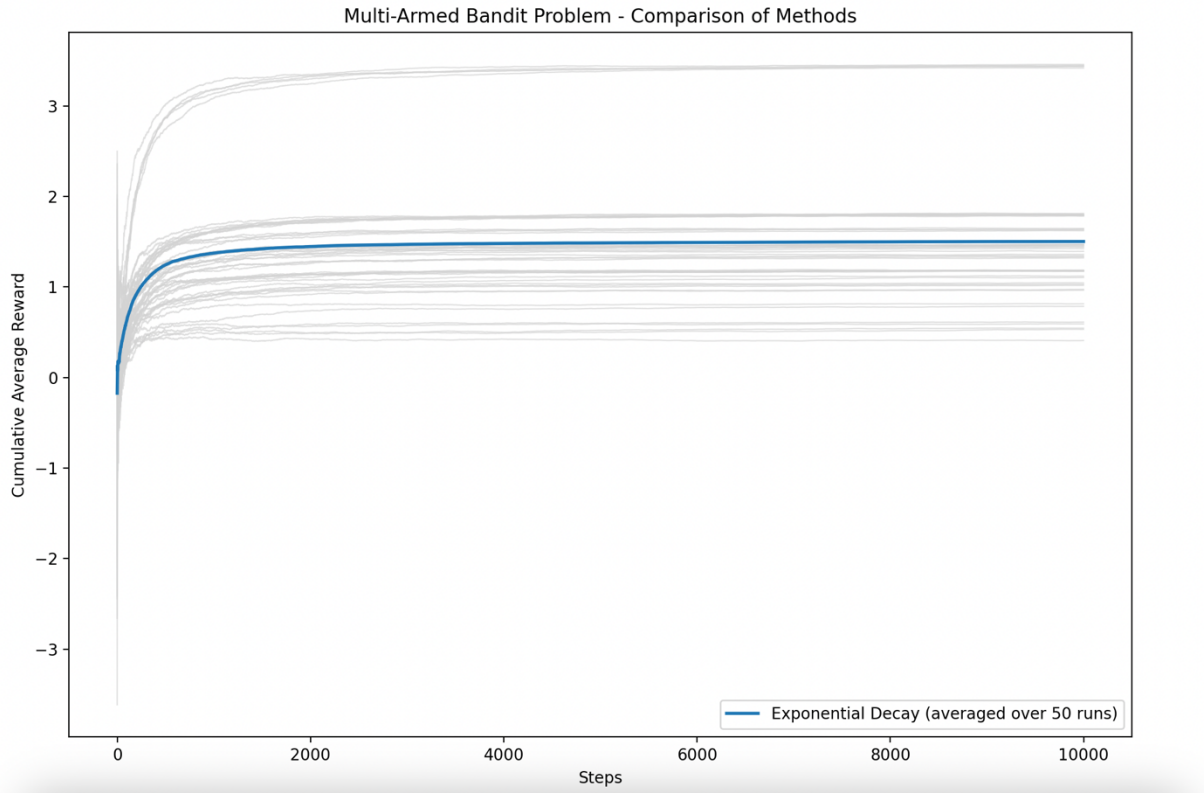
Appendix 28

Linear vs Exponential Decay output (value comparison)



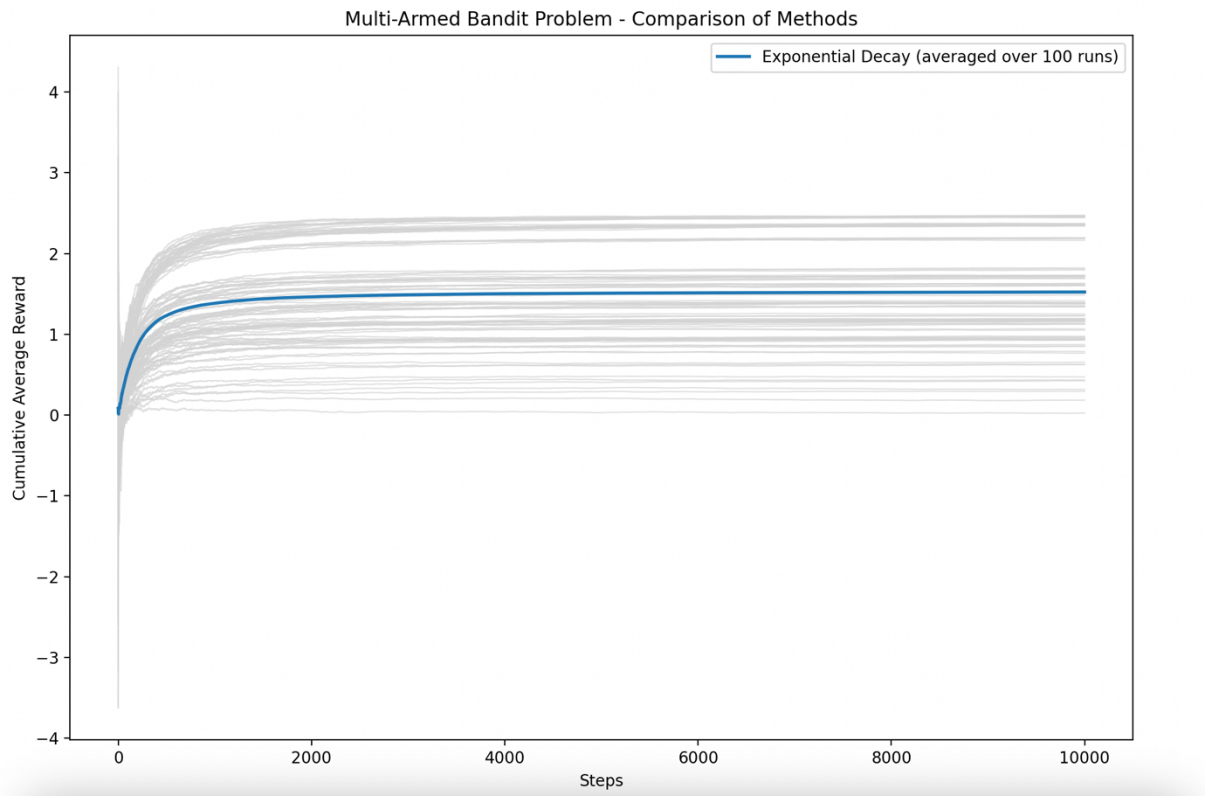
Appendix 29

Exponential Decay output (50 repeats, consistency test)



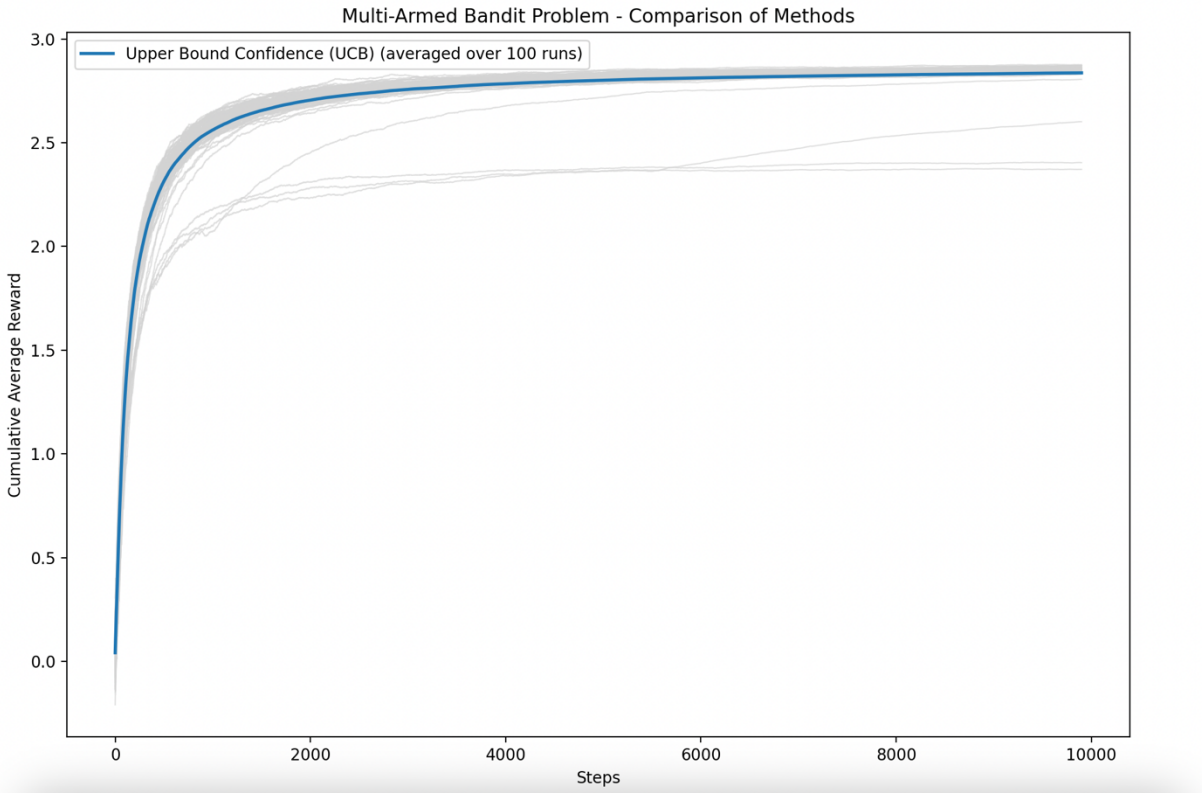
Appendix 30

Exponential Decay output (100 repeats, consistency test)



Appendix 31

UCB, amazing performance



Appendix 32

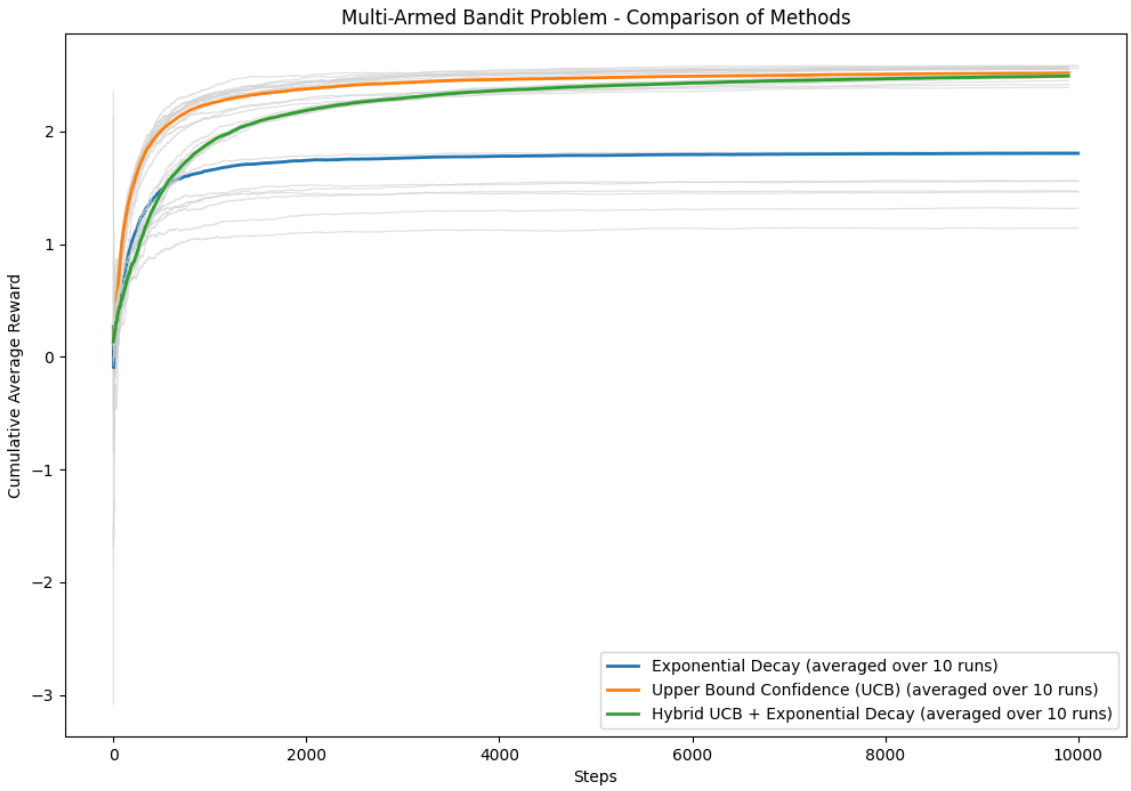
Hybrid function (source code)

```
def hybrid_exponential_ucb(machines, spins, BanditMeans):
    epsilon_0 = 1.0  # Initial exploration rate
    decay_rate = 0.01  # Exponential decay rate
    rewards = [0.0] * machines
    counts = [0] * machines
    acr = []
```

```
    cumulative_reward = 0.0

    # Initialize each machine once (ensuring UCB starts with valid values)
    for i in range(machines):
        reward = np.random.normal(BanditMeans[i], 1)
        rewards[i] += reward
        counts[i] += 1
        cumulative_reward += reward

    # Main loop for action selection
    for i in range(machines, spins):
         epsilon = epsilon_0 * math.exp(-decay_rate * i)  # Exponential decay exploration
rate

        if random.random() < epsilon:
            # Exploration: Choose a random machine
            selected_machine = random.randint(0, machines - 1)
        else:
            # Exploitation: Use UCB without knowing total spins
            upper_bound_all = []
            for j in range(machines):
                avg_reward = rewards[j] / counts[j]
                  exploration_term = math.sqrt((2 * math.log(sum(counts))) / counts[j])  #
Uses past observations only
                upper_bound = avg_reward + exploration_term
                upper_bound_all.append(upper_bound)

            selected_machine = np.argmax(upper_bound_all)

        # Simulate reward
        reward = np.random.normal(BanditMeans[selected_machine], 1)
        rewards[selected_machine] += reward
        counts[selected_machine] += 1
        cumulative_reward += reward

        # Update cumulative average reward
        cumulative_avg_reward = cumulative_reward / (i + 1)
        acr.append(cumulative_avg_reward)

    return acr
```

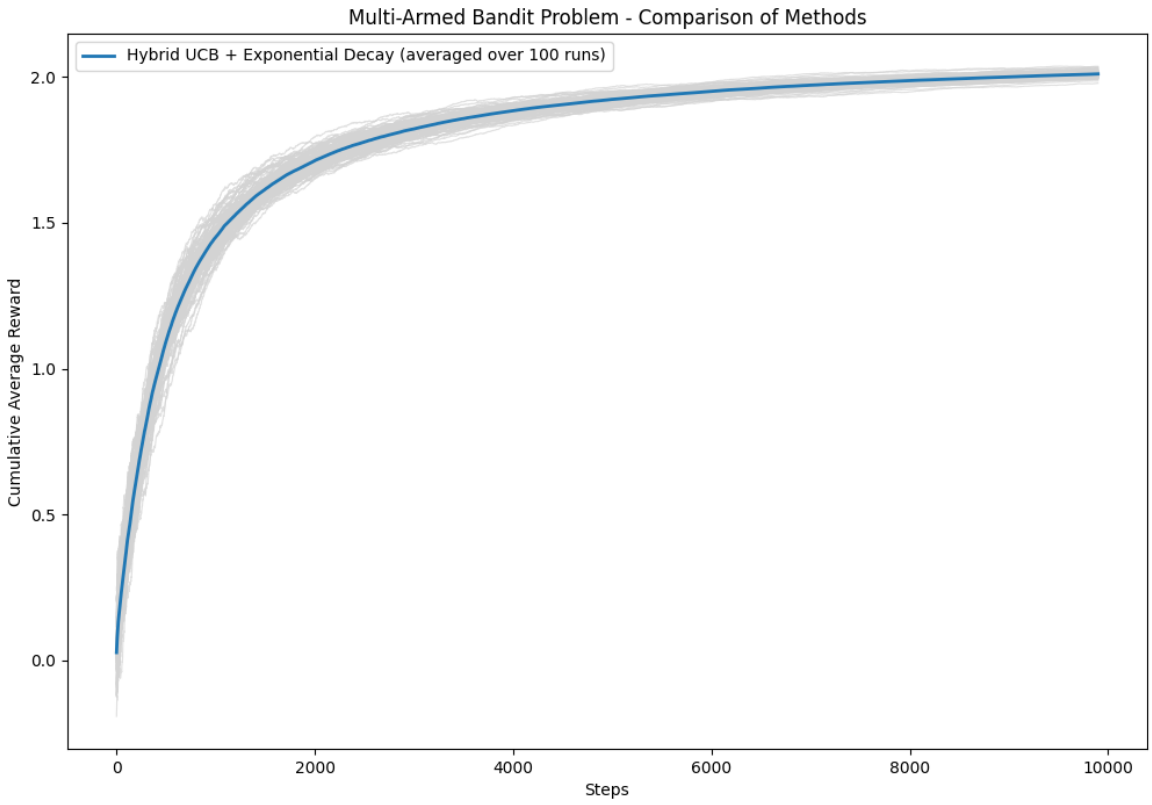
Appendix 33

Exp, UCB, hybrid (comparison, 10 repeats)



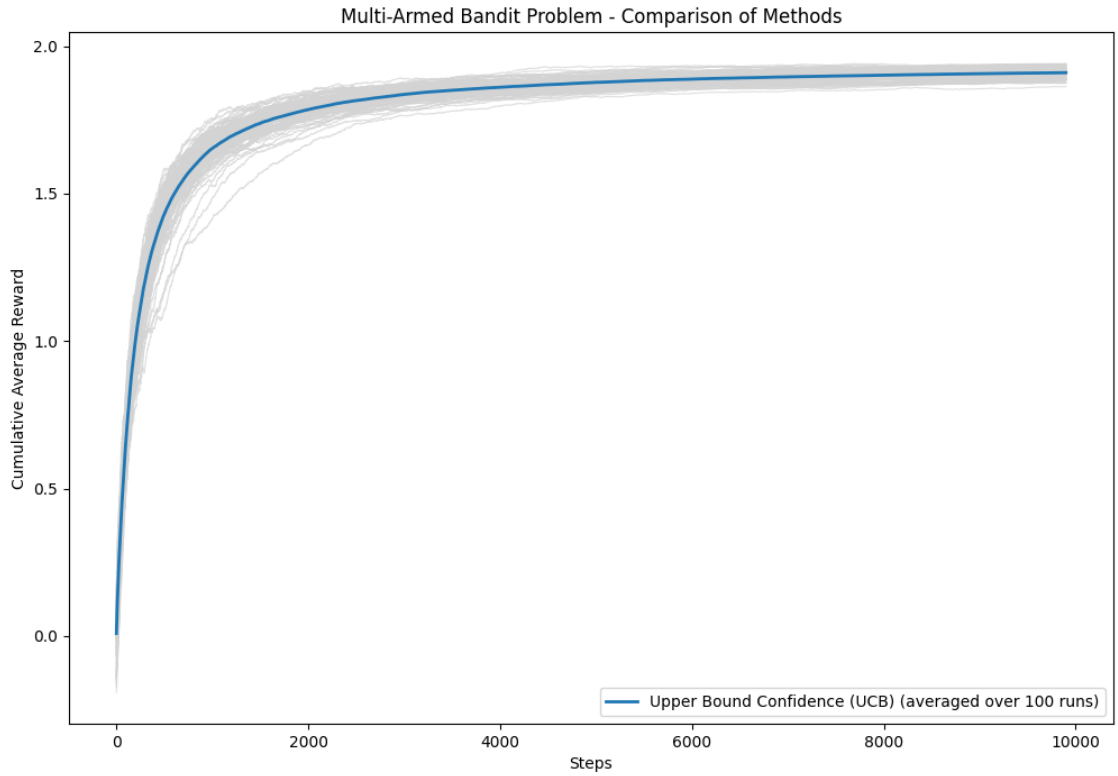
Appendix 34

Hybrid Output (consistency test)



Appendix 35

UBC output (consistency test)



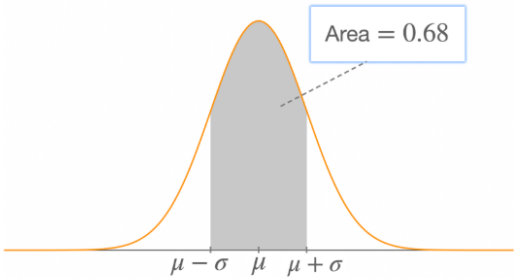
Appendix 36

68, 95, 99.7 heuristic

68% : 1 Standard Deviation, σ

68% of the results fall within 1 standard deviation of the mean μ; that's between 1 standard deviation to the left of the mean and 1 standard deviation to the right of it.

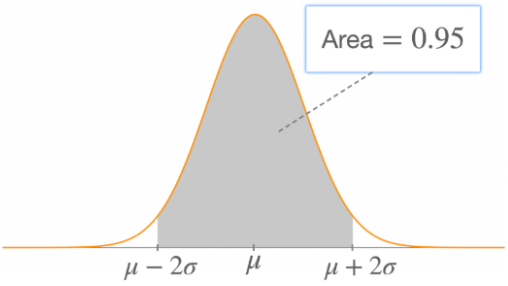
Example: in the case of $\mu = 175$cm and $\sigma = 7$cm, we can state that the probability that a man taken at random measures between 168cm and 182cm is 0.68.



95% : 2 Standard Deviations, 2σ

95% of the results fall within 2 standard deviations of the mean μ.

Example: in the case of $\mu = 175$cm and $\sigma = 7$cm, we can state that the probability that a man taken at random measures between 161cm and 189cm is 0.95.

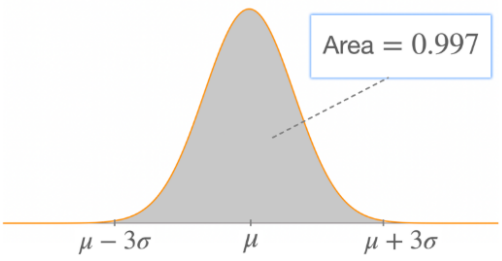


99.7% : 3 Standard Deviations, 3σ

99.7% of the results fall within 3 standard deviations of the mean μ.

Example: in the case of $\mu = 175$cm and $\sigma = 7$cm, we can state that the probability that a man taken at random measures between 154cm and 196cm is 0.997.

As such we can expect 99.7% of the population to measure between 154cm and 196cm tall.



https://www.radfordmathematics.com/probabilities-and-statistics/normal-distributions/68-95-99.7-rule.html

Appendix 37

Randomness testing (source code)

```
import random

numbers = list(range(1, 101))
generated = [random.randint(1, 100) for _ in range(100)]


print("Generated numbers:")
print(generated)


missing_numbers = set(numbers) - set(generated)

print(f"\n{len(missing_numbers)} numbers were not generated:")
print(sorted(missing_numbers))
```

Appendix 38

Randomness testing (output, 100 repeats)

Generated numbers:

[22, 81, 13, 88, 11, 74, 20, 84, 18, 27, 34, 4, 73, 18, 52, 20, 50, 3, 85, 11, 90, 91, 32, 11, 22, 26, 29, 76, 65, 20, 100, 59, 41, 37, 44, 33, 62, 43, 84, 5, 42, 4, 13, 4, 8, 16, 51, 42, 68, 39, 9, 9, 89, 26, 32, 96, 42, 64, 99, 62, 22, 44, 37, 99, 73, 60, 10, 84, 65, 83, 36, 45, 16, 53, 18, 17, 83, 43, 23, 96, 96, 16, 28, 36, 99, 63, 62, 8, 55, 10, 36, 77, 58, 39, 80, 66, 10, 90, 17, 82]

40 numbers were not generated:

[1, 2, 6, 7, 12, 14, 15, 19, 21, 24, 25, 30, 31, 35, 38, 40, 46, 47, 48, 49, 54, 56, 57, 61, 67, 69, 70, 71, 72, 75, 78, 79, 86, 87, 92, 93, 94, 95, 97, 98]

Generated numbers:

[48, 83, 14, 72, 84, 66, 82, 88, 68, 72, 19, 28, 53, 82, 55, 78, 16, 70, 97, 2, 45, 46, 71, 7, 55, 72, 62, 27, 14, 50, 40, 17, 9, 38, 77, 62, 72, 25, 68, 81, 23, 90, 18, 14, 81, 11, 28, 43, 83, 39, 70, 18, 71, 34, 61, 24, 96, 97, 37, 79, 71, 55, 26, 55, 86, 40, 78, 95, 25, 55, 70, 87, 84, 70, 71, 95, 53, 22, 49, 64, 81, 85, 22, 18, 20, 45, 51, 64, 36, 46, 11, 7, 1, 16, 60, 87, 73, 60, 97, 66]

42 numbers were not generated:

[3, 4, 5, 6, 8, 10, 12, 13, 15, 21, 29, 30, 31, 32, 33, 35, 41, 42, 44, 47, 52, 54, 56, 57, 58, 59, 63, 65, 67, 69, 74, 75, 76, 80, 89, 91, 92, 93, 94, 98, 99, 100]

Generated numbers:

[50, 50, 45, 15, 98, 61, 22, 29, 20, 98, 49, 19, 96, 76, 55, 46, 59, 60, 25, 40, 92, 34, 32, 100, 94, 6, 7, 57, 26, 79, 22, 13, 69, 69, 49, 48, 4, 96, 90, 41, 96, 29, 96, 16, 16, 75, 100, 78, 77, 26, 43, 4, 64, 12, 85, 82, 28, 29, 56, 44, 33, 100, 34, 74, 32, 29, 99, 63, 77, 33, 84, 31, 6, 10, 13, 64, 10, 52, 24, 92, 68, 43, 22, 13, 2, 65, 47, 30, 16, 87, 26, 34, 55, 58, 8, 2, 68, 72, 3, 55]

35 numbers were not generated:

[1, 5, 9, 11, 14, 17, 18, 21, 23, 27, 35, 36, 37, 38, 39, 42, 51, 53, 54, 62, 66, 67, 70, 71, 73, 80, 81, 83, 86, 88, 89, 91, 93, 95, 97]

Generated numbers:

[16, 38, 76, 28, 90, 14, 78, 56, 41, 20, 26, 90, 45, 97, 97, 22, 51, 10, 34, 69, 55, 8, 91, 96, 85, 76, 22, 45, 89, 14, 36, 100, 77, 34, 23, 50, 47, 94, 59, 83, 60, 23, 20, 71, 99, 76, 75, 19, 19, 68, 38, 51, 8, 72, 55, 98, 85, 94, 21, 72, 33, 1, 88, 56, 19, 67, 21, 49, 10, 96, 60, 22, 31, 9, 30, 23, 73, 11, 72, 11, 56, 51, 16, 10, 80, 56, 77, 64, 62, 66, 69, 20, 14, 84, 43, 13, 1, 78, 73, 59]

41 numbers were not generated:

[2, 3, 4, 5, 6, 7, 12, 15, 17, 18, 24, 25, 27, 29, 32, 35, 37, 39, 40, 42, 44, 46, 48, 52, 53, 54, 57, 58, 61, 63, 65, 70, 74, 79, 81, 82, 86, 87, 92, 93, 95]

Generated numbers:

[94, 38, 68, 29, 20, 11, 53, 51, 16, 36, 99, 23, 6, 1, 33, 35, 31, 53, 45, 100, 85, 70, 6, 85, 61, 93, 69, 99, 20, 23, 61, 9, 58, 5, 17, 48, 4, 41, 67, 97, 91, 44, 2, 43, 88, 66, 45, 18, 100, 38, 14, 17, 94, 82, 2, 87, 81, 11, 12, 55, 86, 32, 27, 9, 55, 16, 83, 58, 66, 66, 88, 97, 18, 83, 62, 25, 95, 77, 91, 16, 6, 83, 54, 5, 83, 48, 82, 98, 41, 71, 94, 1, 33, 82, 57, 75, 80, 35, 94, 43]

40 numbers were not generated:

[3, 7, 8, 10, 13, 15, 19, 21, 22, 24, 26, 28, 30, 34, 37, 39, 40, 42, 46, 47, 49, 50, 52, 56, 59, 60, 63, 64, 65, 72, 73, 74, 76, 78, 79, 84, 89, 90, 92, 96]

Generated numbers:

[92, 58, 88, 52, 23, 31, 61, 93, 2, 13, 35, 7, 48, 86, 10, 95, 27, 92, 15, 45, 79, 43, 51, 58, 23, 35, 32, 26, 88, 29, 77, 41, 61, 2, 70, 59, 44, 5, 53, 23, 2, 2, 86, 57, 35, 61, 33, 55, 98, 35, 70, 54, 16, 68, 57, 33, 48, 31, 22, 96, 34, 29, 30, 83, 95, 75, 82, 93, 31, 44, 2, 38, 18, 49, 22, 17, 1, 84, 36, 17, 79, 23, 8, 12, 2, 77, 16, 49, 33, 65, 62, 82, 99, 70, 98, 89, 62, 92, 50, 80]

39 numbers were not generated:

[3, 4, 6, 9, 11, 14, 19, 20, 21, 24, 25, 28, 37, 39, 40, 42, 46, 47, 56, 60, 63, 64, 66, 67, 69, 71, 72, 73, 74, 76, 78, 81, 85, 87, 90, 91, 94, 97, 100]

Generated numbers:

[49, 46, 25, 36, 29, 37, 18, 39, 74, 60, 11, 21, 50, 61, 13, 36, 63, 13, 85, 16, 86, 46, 13, 44, 72, 77, 82, 52, 7, 67, 13, 24, 84, 66, 20, 81, 92, 56, 52, 8, 90, 3, 9, 12, 61, 26, 79, 53, 3, 13, 44, 12, 33, 69, 15, 99, 100, 42, 77, 68, 51, 78, 61, 62, 5, 36, 66, 65, 47, 56, 18, 73, 36, 10, 17, 79, 29, 57, 96, 33, 48, 35, 82, 25, 3, 100, 23, 41, 47, 95, 7, 75, 66, 37, 72, 54, 1, 78, 30, 44]

32 numbers were not generated:

[2, 4, 6, 14, 19, 22, 27, 28, 31, 32, 34, 38, 40, 43, 45, 55, 58, 59, 64, 70, 71, 76, 80, 83, 87, 88, 89, 91, 93, 94, 97, 98]

Generated numbers:

[53, 5, 19, 30, 7, 34, 70, 68, 71, 21, 3, 13, 16, 36, 25, 37, 35, 42, 98, 75, 8, 74, 28, 42, 8, 66, 51, 94, 69, 85, 59, 43, 47, 90, 71, 68, 51, 37, 83, 47, 20, 37, 81, 39, 59, 75, 46, 39, 23, 63, 16, 65, 43, 6, 51, 56, 27, 18, 90, 51, 61, 44, 96, 93, 68, 30, 20, 51, 80, 1, 86, 48, 62, 56, 24, 87, 8, 72, 80, 17, 86, 17, 85, 13, 2, 13, 11, 81, 8, 52, 64, 91, 68, 81, 48, 23, 87, 34, 54, 61]

37 numbers were not generated:

[4, 9, 10, 12, 14, 15, 22, 26, 29, 31, 32, 33, 38, 40, 41, 45, 49, 50, 55, 57, 58, 60, 67, 73, 76, 77, 78, 79, 82, 84, 88, 89, 92, 95, 97, 99, 100]

Generated numbers:

[38, 37, 25, 58, 97, 61, 98, 80, 19, 89, 30, 56, 41, 9, 58, 75, 3, 1, 51, 32, 24, 90, 86, 50, 51, 21, 98, 41, 32, 24, 39, 86, 34, 5, 31, 66, 13, 4, 58, 5, 75, 74, 37, 29, 46, 80, 70, 53, 14, 33, 63, 71, 57, 7, 70, 17, 80, 90, 38, 25, 98, 40, 46, 8, 28, 35, 43, 55, 80, 25,

45, 74, 20, 83, 28, 98, 13, 44, 56, 63, 44, 16, 87, 5, 34, 8, 31, 94, 73, 77, 26, 56, 68, 51, 8, 74, 59, 27, 57, 81]

37 numbers were not generated:

[2, 6, 10, 11, 12, 15, 18, 22, 23, 36, 42, 47, 48, 49, 52, 54, 60, 62, 64, 65, 67, 69, 72, 76, 78, 79, 82, 84, 85, 88, 91, 92, 93, 95, 96, 99, 100]

Generated numbers:

[4, 28, 9, 72, 73, 94, 60, 85, 41, 13, 96, 23, 51, 24, 56, 30, 39, 56, 96, 74, 71, 56, 23, 59, 2, 27, 11, 100, 48, 52, 43, 1, 23, 41, 4, 5, 39, 79, 58, 31, 5, 39, 60, 6, 73, 29, 63, 34, 20, 23, 7, 12, 46, 97, 18, 97, 76, 17, 9, 71, 48, 49, 90, 71, 53, 85, 19, 63, 9, 72, 24, 55, 45, 75, 18, 37, 29, 67, 91, 18, 78, 71, 39, 63, 61, 56, 42, 55, 98, 94, 4, 72, 16, 67, 72, 28, 97, 48, 37, 27]

41 numbers were not generated:

[3, 8, 10, 14, 15, 21, 22, 25, 26, 32, 33, 35, 36, 38, 40, 44, 47, 50, 54, 57, 62, 64, 65, 66, 68, 69, 70, 77, 80, 81, 82, 83, 84, 86, 87, 88, 89, 92, 93, 95, 99]

Generated numbers:

[6, 44, 7, 98, 86, 56, 3, 96, 98, 64, 73, 55, 13, 38, 94, 52, 26, 59, 20, 68, 18, 62, 41, 71, 76, 19, 2, 75, 9, 2, 19, 38, 72, 41, 21, 56, 30, 9, 24, 27, 95, 85, 19, 31, 26, 66, 94, 76, 60, 79, 24, 37, 83, 62, 69, 16, 50, 76, 66, 13, 53, 96, 1, 99, 24, 28, 38, 38, 20, 58,

68, 24, 39, 50, 3, 11, 20, 68, 4, 5, 69, 50, 100, 79, 33, 32, 17, 30, 83, 78, 8, 46, 84, 41, 64, 13, 22, 4, 21, 7]

38 numbers were not generated:

[10, 12, 14, 15, 23, 25, 29, 34, 35, 36, 40, 42, 43, 45, 47, 48, 49, 51, 54, 57, 61, 63, 65, 67, 70, 74, 77, 80, 81, 82, 87, 88, 89, 90, 91, 92, 93, 97]

Generated numbers:

[56, 68, 42, 22, 12, 95, 40, 70, 86, 32, 65, 66, 63, 81, 26, 76, 33, 98, 56, 88, 66, 90, 73, 66, 6, 28, 51, 30, 61, 41, 25, 11, 24, 89, 95, 68, 43, 55, 97, 37, 2, 57, 90, 66, 50, 94, 74, 28, 23, 17, 2, 67, 61, 23, 81, 36, 82, 54, 8, 54, 37, 28, 90, 44, 19, 42, 67, 63, 34,

35, 52, 92, 30, 71, 34, 19, 61, 83, 87, 84, 19, 100, 74, 80, 96, 7, 22, 74, 74, 19, 24, 93, 38, 18, 67, 82, 38, 83, 99, 50]

35 numbers were not generated:

[1, 3, 4, 5, 9, 10, 13, 14, 15, 16, 20, 21, 27, 29, 31, 39, 45, 46, 47, 48, 49, 53, 58, 59, 60, 62, 64, 69, 72, 75, 77, 78, 79, 85, 91]

Generated numbers:

[37, 17, 6, 91, 25, 59, 4, 49, 59, 30, 73, 10, 23, 60, 23, 73, 73, 28, 30, 29, 25, 42, 43, 75, 44, 12, 2, 11, 22, 3, 57, 17, 72, 32, 3, 92, 23, 25, 75, 33, 34, 16, 92, 11, 39, 29, 90, 5, 33, 54, 39, 25, 6, 79, 20, 28, 72, 40, 53, 41, 100, 59, 60, 22, 59, 22, 35, 4, 78, 34, 38, 43, 47, 90, 71, 96, 9, 3, 49, 93, 79, 32, 84, 41, 4, 83, 77, 74, 5, 37, 46, 59, 16, 93, 14, 35, 37, 22, 68, 30]

44 numbers were not generated:

[1, 7, 8, 13, 15, 18, 19, 21, 24, 26, 27, 31, 36, 45, 48, 50, 51, 52, 55, 56, 58, 61, 62, 63, 64, 65, 66, 67, 69, 70, 76, 80, 81, 82, 85, 86, 87, 88, 89, 94, 95, 97, 98, 99]

Generated numbers:

[44, 15, 60, 92, 2, 93, 56, 82, 2, 41, 98, 8, 64, 85, 53, 96, 97, 55, 41, 100, 33, 12, 75, 72, 83, 46, 18, 9, 99, 72, 20, 90, 77, 75, 83, 23, 53, 17, 51, 47, 76, 40, 46, 93, 87, 79, 30, 85, 99, 23, 46, 37, 14, 56, 100, 19, 44, 38, 48, 2, 64, 26, 87, 97, 13, 87, 76, 59, 53, 1, 16, 96, 20, 86, 65, 26, 1, 83, 74, 60, 53, 49, 20, 53, 84, 34, 61, 89, 49, 47, 34, 26, 67, 70, 61, 65, 16, 66, 13, 95]

39 numbers were not generated:

[3, 4, 5, 6, 7, 10, 11, 21, 22, 24, 25, 27, 28, 29, 31, 32, 35, 36, 39, 42, 43, 45, 50, 52, 54, 57, 58, 62, 63, 68, 69, 71, 73, 78, 80, 81, 88, 91, 94]

Generated numbers:

[88, 69, 22, 87, 44, 80, 47, 26, 35, 90, 32, 66, 46, 68, 3, 24, 42, 55, 74, 55, 72, 31, 76, 74, 3, 69, 41, 13, 17, 24, 38, 11, 61, 57, 67, 26, 23, 23, 70, 52, 16, 2, 6, 89, 49, 79, 29, 96, 44, 46, 1, 22, 51, 52, 32, 86, 71, 69, 16, 16, 70, 25, 27, 98, 9, 25, 68, 100, 2, 60, 83, 49, 38, 48, 90, 95, 73, 99, 96, 50, 80, 54, 88, 82, 30, 64, 20, 9, 93, 53, 8, 79, 68, 3, 91, 48, 92, 77, 7, 97]

30 numbers were not generated:

[4, 5, 10, 12, 14, 15, 18, 19, 21, 28, 33, 34, 36, 37, 39, 40, 43, 45, 56, 58, 59, 62, 63, 65, 75, 78, 81, 84, 85, 94]

Generated numbers:

[17, 39, 71, 78, 60, 61, 24, 92, 11, 73, 65, 52, 78, 89, 65, 16, 41, 59, 26, 24, 84, 98, 36, 79, 37, 71, 97, 69, 4, 88, 91, 12, 100, 52, 69, 40, 16, 15, 36, 35, 7, 73, 51, 33, 59, 38, 98, 89, 23, 19, 59, 90, 90, 31, 99, 13, 60, 29, 49, 74, 42, 48, 60, 33, 77, 85, 1, 59, 98, 76, 23, 93, 70, 71, 90, 81, 41, 33, 87, 15, 20, 48, 87, 23, 94, 51, 32, 43, 34, 32, 23, 17, 71, 84, 7, 39, 78, 37, 2, 6]

38 numbers were not generated:

[3, 5, 8, 9, 10, 14, 18, 21, 22, 25, 27, 28, 30, 44, 45, 46, 47, 50, 53, 54, 55, 56, 57, 58, 62, 63, 64, 66, 67, 68, 72, 75, 80, 82, 83, 86, 95, 96]

Generated numbers:

[99, 10, 95, 41, 33, 46, 11, 27, 36, 11, 26, 94, 55, 71, 91, 5, 34, 23, 94, 37, 23, 98, 18, 95, 91, 95, 61, 13, 22, 71, 90, 86, 26, 78, 100, 79, 6, 94, 31, 100, 62, 71, 75, 100, 29, 82, 24, 8, 29, 57, 4, 86, 48, 9, 7, 9, 11, 60, 25, 54, 39, 59, 94, 59, 21, 44, 40, 55, 98,

84, 13, 51, 73, 81, 90, 54, 87, 37, 11, 3, 25, 45, 33, 11, 53, 69, 19, 28, 35, 36, 44, 77, 90, 74, 8, 55, 80, 86, 6, 56]

35 numbers were not generated:

[1, 2, 12, 14, 15, 16, 17, 20, 30, 32, 38, 42, 43, 47, 49, 50, 52, 58, 63, 64, 65, 66, 67, 68, 70, 72, 76, 83, 85, 88, 89, 92, 93, 96, 97]

Generated numbers:

[22, 80, 76, 18, 100, 91, 12, 69, 78, 43, 14, 69, 30, 6, 99, 33, 69, 70, 90, 43, 65, 50, 51, 60, 42, 14, 37, 8, 99, 21, 86, 17, 92, 46, 60, 24, 95, 27, 5, 38, 33, 91, 40, 48, 82, 86, 1, 24, 54, 22, 73, 41, 69, 47, 49, 8, 42, 98, 44, 15, 42, 36, 94, 50, 45, 44, 22, 25, 26,

17, 69, 35, 77, 98, 98, 38, 98, 61, 51, 29, 9, 63, 35, 73, 48, 56, 47, 67, 6, 87, 42, 21, 5, 79, 84, 63, 58, 100, 97, 2]

35 numbers were not generated:

[3, 4, 7, 10, 11, 13, 16, 19, 20, 23, 28, 31, 32, 34, 39, 52, 53, 55, 57, 59, 62, 64, 66, 68, 71, 72, 74, 75, 81, 83, 85, 88, 89, 93, 96]

Generated numbers:

[24, 73, 31, 70, 16, 52, 82, 97, 33, 22, 25, 65, 24, 71, 17, 48, 7, 17, 25, 69, 4, 52, 70, 6, 50, 49, 81, 39, 99, 70, 52, 100, 81, 12, 45, 22, 79, 57, 3, 80, 21, 71, 25, 49, 42, 82, 4, 80, 51, 6, 91, 6, 4, 32, 34, 49, 35, 46, 59, 54, 21, 30, 36, 74, 70, 71, 67, 66, 90, 18, 36, 27, 26, 82, 11, 16, 19, 90, 28, 13, 40, 93, 86, 17, 96, 49, 34, 64, 82, 48, 61, 36, 61, 63, 69, 82, 30, 45, 42, 56]

38 numbers were not generated:

[1, 2, 5, 8, 9, 10, 14, 15, 20, 23, 29, 37, 38, 41, 43, 44, 47, 53, 55, 58, 60, 62, 68, 72, 75, 76, 77, 78, 83, 84, 85, 87, 88, 89, 92, 94, 95, 98]

Generated numbers:

[55, 8, 77, 58, 95, 56, 86, 27, 10, 86, 37, 32, 91, 21, 38, 31, 27, 38, 58, 93, 65, 9, 97, 48, 30, 36, 29, 30, 62, 88, 91, 89, 43, 57, 24, 81, 95, 8, 98, 22, 14, 54, 41, 22, 35, 36, 76, 45, 43, 98, 69, 10, 34, 69, 79, 46, 97, 36, 27, 29, 98, 68, 64, 30, 11, 74, 56, 65, 44, 99, 47, 53, 59, 79, 90, 64, 81, 52, 77, 94, 24, 18, 90, 1, 68, 81, 45, 46, 76, 6, 23, 19, 30, 4, 91, 40, 13, 65, 93, 9]

38 numbers were not generated:

[2, 3, 5, 7, 12, 15, 16, 17, 20, 25, 26, 28, 33, 39, 42, 49, 50, 51, 60, 61, 63, 66, 67, 70, 71, 72, 73, 75, 78, 80, 82, 83, 84, 85, 87, 92, 96, 100]

Generated numbers:

[42, 8, 71, 92, 46, 89, 17, 85, 79, 12, 40, 98, 89, 15, 58, 55, 42, 20, 100, 31, 55, 100, 98, 29, 75, 47, 55, 85, 30, 37, 100, 89, 4, 63, 75, 100, 35, 17, 35, 55, 68, 24, 71, 46, 90, 86, 72, 43, 9, 89, 69, 44, 28, 70, 58, 42, 37, 6, 52, 30, 65, 29, 72, 59, 97, 74, 50, 84,

46, 15, 88, 96, 75, 96, 24, 79, 29, 59, 84, 93, 82, 87, 55, 57, 42, 1, 4, 51, 24, 41, 84, 61, 38, 70, 34, 55, 1, 81, 91, 62]

40 numbers were not generated:

[2, 3, 5, 7, 10, 11, 13, 14, 16, 18, 19, 21, 22, 23, 25, 26, 27, 32, 33, 36, 39, 45, 48, 49, 53, 54, 56, 60, 64, 66, 67, 73, 76, 77, 78, 80, 83, 94, 95, 99]

Generated numbers:

[42, 74, 69, 22, 88, 83, 31, 25, 47, 93, 64, 91, 49, 92, 14, 18, 50, 64, 81, 48, 12, 80, 6, 18, 44, 35, 2, 29, 72, 60, 68, 46, 27, 59, 96, 97, 97, 44, 32, 77, 12, 36, 54, 6, 96, 38, 6, 60, 76, 78, 65, 58, 100, 10, 75, 68, 22, 76, 61, 60, 98, 41, 21, 49, 92, 73, 44, 76, 16, 61, 67, 98, 37, 93, 60, 61, 97, 66, 52, 65, 17, 53, 35, 8, 39, 20, 38, 71, 56, 94, 66, 49, 18, 93, 25, 22, 73, 85, 8, 91]

35 numbers were not generated:

[1, 3, 4, 5, 7, 9, 11, 13, 15, 19, 23, 24, 26, 28, 30, 33, 34, 40, 43, 45, 51, 55, 57, 62, 63, 70, 79, 82, 84, 86, 87, 89, 90, 95, 99]

Generated numbers:

[47, 56, 27, 43, 85, 59, 100, 55, 100, 30, 21, 8, 92, 28, 69, 74, 94, 64, 63, 91, 22, 16, 20, 15, 79, 33, 40, 24, 58, 8, 9, 93, 8, 57, 27, 29, 71, 7, 44, 52, 50, 67, 39, 18, 86, 73, 56, 17, 84, 67, 62, 34, 11, 41, 1, 6, 64, 5, 16, 80, 72, 48, 62, 18, 14, 11, 29, 82, 75, 28, 56, 86, 42, 75, 72, 97, 98, 35, 50, 67, 5, 11, 90, 14, 91, 61, 55, 48, 94, 58, 84, 27, 23, 97, 52, 43, 42, 73, 73, 62]

36 numbers were not generated:

[2, 3, 4, 10, 12, 13, 19, 25, 26, 31, 32, 36, 37, 38, 45, 46, 49, 51, 53, 54, 60, 65, 66, 68, 70, 76, 77, 78, 81, 83, 87, 88, 89, 95, 96, 99]

Generated numbers:

[90, 78, 54, 61, 7, 59, 70, 52, 56, 19, 5, 56, 49, 32, 57, 1, 84, 45, 54, 99, 37, 94, 55, 9, 98, 77, 49, 4, 44, 68, 11, 59, 62, 22, 36, 67, 84, 51, 4, 11, 34, 42, 92, 96, 54, 29, 66, 82, 55, 5, 14, 16, 8, 18, 12, 85, 73, 4, 20, 98, 50, 3, 38, 30, 96, 82, 85, 15, 62, 47, 86, 31, 86, 86, 87, 13, 93, 92, 12, 92, 59, 56, 23, 23, 61, 79, 68, 81, 44, 76, 34, 81, 13, 46, 37, 72, 76, 83, 25, 99]

33 numbers were not generated:

[2, 6, 10, 17, 21, 24, 26, 27, 28, 33, 35, 39, 40, 41, 43, 48, 53, 58, 60, 63, 64, 65, 69, 71, 74, 75, 80, 88, 89, 91, 95, 97, 100]

Generated numbers:

[47, 75, 51, 59, 100, 96, 100, 94, 64, 49, 54, 78, 37, 16, 60, 23, 33, 40, 21, 33, 3, 1, 98, 1, 18, 79, 49, 49, 48, 17, 47, 7, 87, 19, 12, 24, 27, 61, 48, 68, 26, 81, 97, 50, 36, 4, 25, 41, 81, 71, 12, 42, 34, 15, 57, 7, 34, 51, 36, 77, 72, 9, 43, 50, 77, 41, 95, 76, 79, 11, 11, 96, 47, 46, 94, 14, 51, 76, 73, 52, 85, 70, 42, 25, 86, 92, 73, 65, 65, 8, 41, 71, 24, 53, 1, 15, 91, 56, 83, 52]

33 numbers were not generated:

[2, 5, 6, 10, 13, 20, 22, 28, 29, 30, 31, 32, 35, 38, 39, 44, 45, 55, 58, 62, 63, 66, 67, 69, 74, 80, 82, 84, 88, 89, 90, 93, 99]

Generated numbers:

[58, 36, 44, 41, 61, 72, 93, 43, 88, 17, 50, 68, 38, 31, 93, 71, 68, 68, 77, 2, 9, 19, 25, 93, 23, 32, 72, 31, 21, 98, 2, 70, 86, 4, 64, 67, 81, 92, 84, 97, 74, 89, 74, 61, 32, 45, 31, 80, 24, 22, 37, 66, 84, 44, 61, 98, 87, 79, 48, 87, 48, 83, 45, 34, 57, 17, 14, 2, 6, 56, 75, 21, 44, 45, 1, 18, 95, 84, 9, 56, 68, 9, 42, 94, 45, 4, 50, 16, 77, 57, 36, 80, 9, 51, 26, 76, 30, 32, 45, 10]

38 numbers were not generated:

[3, 5, 7, 8, 11, 12, 13, 15, 20, 27, 28, 29, 33, 35, 39, 40, 46, 47, 49, 52, 53, 54, 55, 59, 60, 62, 63, 65, 69, 73, 78, 82, 85, 90, 91, 96, 99, 100]

Generated numbers:

[71, 79, 43, 82, 48, 15, 86, 99, 47, 13, 15, 45, 16, 93, 19, 78, 8, 86, 84, 95, 31, 48, 50, 12, 53, 76, 76, 5, 51, 15, 45, 68, 20, 89, 3, 86, 71, 85, 65, 35, 57, 64, 34, 40, 42, 51, 7, 9, 67, 81, 59, 12, 85, 1, 22, 10, 21, 54, 99, 2, 46, 21, 92, 80, 23, 34, 69, 91, 90, 97, 80, 83, 53, 99, 71, 94, 18, 83, 70, 7, 75, 7, 92, 63, 56, 26, 92, 32, 82, 35, 44, 83, 10, 19, 40, 26, 59, 35, 74, 66]

32 numbers were not generated:

[4, 6, 11, 14, 17, 24, 25, 27, 28, 29, 30, 33, 36, 37, 38, 39, 41, 49, 52, 55, 58, 60, 61, 62, 72, 73, 77, 87, 88, 96, 98, 100]

Generated numbers:

[61, 100, 37, 39, 50, 31, 39, 76, 30, 90, 47, 82, 45, 6, 87, 60, 55, 71, 69, 2, 65, 90, 12, 67, 2, 82, 2, 56, 18, 47, 51, 20, 95, 58, 63, 41, 96, 33, 85, 10, 94, 68, 70, 49, 17, 48, 100, 63, 31, 35, 17, 47, 58, 95, 77, 35, 86, 44, 51, 99, 54, 94, 9, 21, 100, 81, 25, 59, 71, 33, 1, 14, 46, 36, 77, 28, 23, 53, 33, 21, 79, 65, 31, 94, 22, 51, 78, 7, 54, 12, 80, 56, 53, 63, 20, 27, 49, 47, 89, 77]

35 numbers were not generated:

[3, 4, 5, 8, 11, 13, 15, 16, 19, 24, 26, 29, 32, 34, 38, 40, 42, 43, 52, 57, 62, 64, 66, 72, 73, 74, 75, 83, 84, 88, 91, 92, 93, 97, 98]

Generated numbers:

[44, 76, 74, 92, 78, 65, 23, 29, 17, 38, 80, 99, 49, 22, 24, 67, 89, 47, 32, 6, 7, 11, 81, 7, 76, 51, 61, 24, 59, 20, 87, 54, 78, 60, 58, 30, 6, 45, 99, 98, 22, 11, 48, 77, 52, 12, 5, 30, 46, 73, 62, 36, 22, 8, 5, 81, 76, 22, 52, 12, 3, 22, 55, 20, 8, 71, 84, 21, 88, 47, 95, 83, 55, 70, 49, 20, 65, 91, 43, 26, 35, 19, 63, 8, 43, 80, 31, 47, 26, 35, 98, 16, 76, 6, 8, 88, 31, 53, 55, 100]

37 numbers were not generated:

[1, 2, 4, 9, 10, 13, 14, 15, 18, 25, 27, 28, 33, 34, 37, 39, 40, 41, 42, 50, 56, 57, 64, 66, 68, 69, 72, 75, 79, 82, 85, 86, 90, 93, 94, 96, 97]

Generated numbers:

[52, 36, 11, 44, 88, 69, 65, 25, 71, 11, 3, 68, 87, 33, 23, 76, 3, 68, 92, 90, 89, 48, 85, 65, 54, 26, 82, 16, 86, 17, 100, 27, 19, 69, 79, 88, 63, 50, 68, 61, 39, 51, 54, 3, 17, 47, 8, 69, 69, 35, 74, 94, 65, 46, 5, 26, 34, 57, 57, 38, 38, 16, 45, 58, 29, 44, 56, 70, 38,

61, 51, 40, 40, 20, 91, 68, 56, 55, 25, 58, 25, 51, 77, 69, 27, 84, 54, 95, 35, 6, 94, 74, 58, 30, 14, 74, 16, 57, 80, 1]

38 numbers were not generated:

[2, 4, 7, 9, 10, 12, 13, 15, 18, 21, 22, 24, 28, 31, 32, 37, 41, 42, 43, 49, 53, 59, 60, 62, 64, 66, 67, 72, 73, 75, 78, 81, 83, 93, 96, 97, 98, 99]

Generated numbers:

[51, 29, 84, 30, 20, 34, 99, 26, 32, 73, 65, 76, 86, 83, 60, 23, 44, 13, 66, 2, 42, 15, 43, 25, 72, 70, 91, 11, 56, 44, 37, 62, 7, 1, 65, 100, 75, 30, 71, 2, 31, 14, 19, 58, 24, 56, 42, 40, 95, 37, 74, 8, 59, 33, 42, 86, 66, 27, 82, 60, 26, 40, 21, 74, 95, 37, 74, 42, 62,

69, 52, 63, 1, 93, 33, 43, 83, 37, 77, 66, 8, 41, 90, 4, 5, 67, 78, 40, 73, 93, 49, 27, 89, 35, 36, 18, 57, 30, 72, 56]

33 numbers were not generated:

[3, 6, 9, 10, 12, 16, 17, 22, 28, 38, 39, 45, 46, 47, 48, 50, 53, 54, 55, 61, 64, 68, 79, 80, 81, 85, 87, 88, 92, 94, 96, 97, 98]

Generated numbers:

[50, 61, 43, 51, 36, 23, 44, 87, 16, 39, 82, 45, 60, 52, 48, 44, 92, 36, 63, 50, 11, 54, 28, 28, 17, 22, 62, 3, 17, 22, 81, 52, 93, 59, 73, 31, 12, 53, 49, 22, 4, 10, 32, 83, 87, 14, 19, 14, 90, 25, 3, 45, 39, 37, 22, 87, 39, 1, 58, 79, 96, 99, 66, 42, 74, 20, 1, 58, 13, 42, 13, 90, 1, 98, 8, 70, 27, 21, 90, 73, 69, 30, 38, 18, 63, 95, 56, 73, 83, 88, 30, 95, 38, 51, 91, 21, 64, 70, 29, 39]

34 numbers were not generated:

[2, 5, 6, 7, 9, 15, 24, 26, 33, 34, 35, 40, 41, 46, 47, 55, 57, 65, 67, 68, 71, 72, 75, 76, 77, 78, 80, 84, 85, 86, 89, 94, 97, 100]

Generated numbers:

[62, 3, 64, 30, 5, 30, 89, 59, 47, 25, 53, 58, 42, 7, 4, 18, 93, 85, 82, 64, 29, 4, 52, 80, 91, 51, 1, 85, 7, 52, 42, 77, 54, 10, 64, 2, 2, 52, 96, 56, 76, 63, 94, 40, 55, 10, 15, 48, 72, 26, 61, 69, 26, 34, 86, 93, 36, 65, 93, 84, 5, 79, 58, 33, 61, 44, 98, 52, 97, 57, 5, 30, 63, 79, 21, 25, 72, 32, 24, 50, 91, 54, 9, 37, 77, 99, 48, 21, 11, 64, 2, 42, 89, 97, 30, 44, 85, 93, 8, 10]

39 numbers were not generated:

[6, 12, 13, 14, 16, 17, 19, 20, 22, 23, 27, 28, 31, 35, 38, 39, 41, 43, 45, 46, 49, 60, 66, 67, 68, 70, 71, 73, 74, 75, 78, 81, 83, 87, 88, 90, 92, 95, 100]

Generated numbers:

[75, 86, 70, 40, 72, 63, 53, 53, 60, 76, 26, 62, 89, 74, 77, 70, 94, 49, 59, 3, 50, 40, 60, 1, 10, 39, 63, 25, 27, 69, 13, 34, 93, 59, 44, 54, 48, 92, 71, 56, 17, 55, 3, 78, 79, 54, 8, 65, 85, 97, 87, 10, 53, 40, 47, 42, 68, 41, 39, 41, 65, 31, 18, 69, 92, 66, 45, 98, 70,

64, 100, 82, 83, 45, 22, 47, 39, 3, 48, 25, 63, 39, 49, 69, 34, 91, 29, 59, 46, 35, 36, 23, 62, 86, 84, 76, 57, 91, 46, 34]

35 numbers were not generated:

[2, 4, 5, 6, 7, 9, 11, 12, 14, 15, 16, 19, 20, 21, 24, 28, 30, 32, 33, 37, 38, 43, 51, 52, 58, 61, 67, 73, 80, 81, 88, 90, 95, 96, 99]

Generated numbers:

[76, 49, 30, 44, 1, 94, 52, 23, 24, 22, 61, 77, 45, 1, 100, 84, 45, 3, 69, 34, 44, 80, 25, 52, 69, 43, 18, 34, 13, 86, 25, 9, 46, 26, 71, 86, 78, 1, 70, 20, 46, 29, 69, 91, 63, 76, 73, 99, 75, 3, 78, 77, 77, 47, 31, 41, 12, 72, 55, 90, 29, 89, 80, 21, 68, 97, 55, 77, 26, 95, 13, 97, 16, 84, 41, 97, 27, 59, 90, 19, 60, 77, 99, 30, 44, 17, 72, 59, 25, 26, 81, 91, 16, 38, 83, 76, 86, 37, 25, 56]

40 numbers were not generated:

[2, 4, 5, 6, 7, 8, 10, 11, 14, 15, 28, 32, 33, 35, 36, 39, 40, 42, 48, 50, 51, 53, 54, 57, 58, 62, 64, 65, 66, 67, 74, 79, 82, 85, 87, 88, 92, 93, 96, 98]

Generated numbers:

[89, 95, 80, 96, 70, 48, 70, 50, 44, 71, 98, 87, 85, 64, 54, 18, 20, 7, 7, 49, 7, 29, 58, 46, 60, 89, 95, 49, 5, 95, 4, 38, 89, 99, 47, 64, 62, 81, 47, 71, 72, 53, 73, 29, 45, 12, 56, 8, 17, 11, 98, 22, 43, 76, 95, 96, 95, 98, 7, 85, 26, 15, 54, 26, 5, 76, 42, 38, 57, 68,

85, 74, 26, 82, 29, 86, 95, 86, 47, 38, 79, 51, 97, 49, 78, 83, 29, 45, 27, 5, 25, 46, 37, 64, 61, 60, 22, 52, 10, 48]

40 numbers were not generated:

[1, 2, 3, 6, 9, 13, 14, 16, 19, 21, 23, 24, 28, 30, 31, 32, 33, 34, 35, 36, 39, 40, 41, 55, 59, 63, 65, 66, 67, 69, 75, 77, 84, 88, 90, 91, 92, 93, 94, 100]

Generated numbers:

[2, 31, 96, 3, 2, 25, 33, 62, 79, 27, 97, 46, 52, 20, 20, 70, 66, 51, 28, 81, 34, 63, 94, 12, 28, 57, 33, 71, 71, 99, 32, 32, 17, 87, 40, 39, 65, 90, 14, 48, 45, 61, 57, 71, 72, 72, 6, 7, 82, 63, 24, 96, 45, 15, 28, 25, 73, 73, 83, 18, 80, 9, 9, 44, 91, 25, 13, 12, 88, 100, 63, 73, 42, 8, 54, 42, 93, 32, 12, 30, 2, 85, 68, 5, 83, 76, 54, 26, 6, 61, 49, 66, 57, 82, 77, 4, 79, 94, 91, 59]

34 numbers were not generated:

[1, 10, 11, 16, 19, 21, 22, 23, 29, 35, 36, 37, 38, 41, 43, 47, 50, 53, 55, 56, 58, 60, 64, 67, 69, 74, 75, 78, 84, 86, 89, 92, 95, 98]

Generated numbers:

[90, 19, 66, 30, 74, 100, 77, 6, 91, 33, 22, 93, 70, 37, 7, 100, 47, 74, 44, 37, 75, 7, 12, 63, 48, 26, 5, 2, 49, 22, 94, 90, 98, 60, 27, 82, 11, 1, 3, 49, 10, 58, 15, 77, 2, 69, 83, 20, 88, 52, 74, 47, 14, 100, 54, 4, 43, 31, 53, 4, 49, 29, 42, 70, 6, 9, 4, 68, 57, 25, 66, 38, 16, 52, 98, 38, 75, 67, 36, 30, 89, 84, 67, 60, 20, 57, 35, 1, 52, 39, 20, 46, 14, 15, 33, 63, 56, 66, 38, 93]

37 numbers were not generated:

[8, 13, 17, 18, 21, 23, 24, 28, 32, 34, 40, 41, 45, 50, 51, 55, 59, 61, 62, 64, 65, 71, 72, 73, 76, 78, 79, 80, 81, 85, 86, 87, 92, 95, 96, 97, 99]

Generated numbers:

[83, 39, 23, 72, 56, 89, 42, 47, 6, 75, 56, 37, 100, 13, 51, 11, 13, 76, 49, 2, 65, 90, 26, 32, 64, 66, 30, 9, 86, 55, 91, 66, 24, 14, 91, 18, 85, 51, 55, 77, 75, 8, 16, 66, 43, 43, 67, 79, 45, 46, 59, 76, 39, 99, 77, 45, 59, 77, 24, 74, 65, 1, 19, 34, 65, 13, 16, 86, 14,

68, 86, 96, 30, 50, 35, 39, 36, 57, 86, 23, 53, 71, 12, 4, 63, 94, 82, 32, 77, 34, 19, 72, 65, 47, 54, 34, 50, 50, 49, 82]

39 numbers were not generated:

[3, 5, 7, 10, 15, 17, 20, 21, 22, 25, 27, 28, 29, 31, 33, 38, 40, 41, 44, 48, 52, 58, 60, 61, 62, 69, 70, 73, 78, 80, 81, 84, 87, 88, 92, 93, 95, 97, 98]

Generated numbers:

[38, 36, 17, 63, 65, 98, 9, 60, 20, 86, 25, 52, 85, 81, 78, 84, 54, 56, 92, 23, 54, 57, 61, 36, 39, 55, 12, 3, 70, 61, 83, 39, 70, 33, 18, 35, 22, 67, 27, 65, 18, 39, 23, 25, 61, 48, 1, 74, 96, 45, 31, 93, 71, 18, 51, 40, 10, 16, 95, 43, 93, 51, 27, 32, 61, 44, 69, 2, 16,

65, 75, 90, 13, 81, 20, 53, 16, 44, 32, 67, 83, 41, 30, 72, 4, 5, 45, 97, 46, 33, 96, 68, 97, 48, 23, 48, 83, 9, 99, 12]

35 numbers were not generated:

[6, 7, 8, 11, 14, 15, 19, 21, 24, 26, 28, 29, 34, 37, 42, 47, 49, 50, 58, 59, 62, 64, 66, 73, 76, 77, 79, 80, 82, 87, 88, 89, 91, 94, 100]

Generated numbers:

[7, 33, 18, 78, 79, 41, 40, 3, 57, 43, 59, 28, 36, 73, 73, 73, 96, 90, 73, 17, 34, 9, 24, 35, 36, 71, 75, 48, 37, 100, 77, 97, 69, 74, 25, 86, 43, 57, 4, 66, 58, 59, 86, 44, 43, 17, 25, 14, 3, 95, 42, 33, 27, 57, 44, 88, 75, 15, 50, 19, 63, 29, 34, 63, 22, 34, 55, 44, 36,

35, 23, 51, 96, 10, 61, 38, 95, 27, 67, 57, 16, 40, 31, 73, 9, 44, 26, 8, 67, 64, 78, 43, 52, 45, 17, 56, 33, 4, 25, 57]

38 numbers were not generated:

[1, 2, 5, 6, 11, 12, 13, 20, 21, 30, 32, 39, 46, 47, 49, 53, 54, 60, 62, 65, 68, 70, 72, 76, 80, 81, 82, 83, 84, 85, 87, 89, 91, 92, 93, 94, 98, 99]

Generated numbers:

[19, 46, 31, 45, 4, 26, 93, 16, 58, 80, 31, 2, 52, 70, 29, 29, 86, 31, 27, 39, 24, 18, 38, 83, 83, 52, 32, 43, 97, 81, 52, 29, 54, 60, 42, 79, 8, 81, 12, 26, 94, 25, 29, 97, 21, 68, 98, 3, 58, 97, 68, 3, 70, 17, 79, 7, 17, 62, 35, 89, 91, 30, 98, 65, 83, 75, 33, 33, 19, 43, 69, 91, 87, 29, 26, 68, 20, 18, 74, 32, 23, 17, 26, 48, 31, 5, 28, 77, 89, 99, 48, 1, 28, 97, 47, 38, 97, 73, 71, 41]

38 numbers were not generated:

[6, 9, 10, 11, 13, 14, 15, 22, 34, 36, 37, 40, 44, 49, 50, 51, 53, 55, 56, 57, 59, 61, 63, 64, 66, 67, 72, 76, 78, 82, 84, 85, 88, 90, 92, 95, 96, 100]

Generated numbers:

[27, 12, 37, 23, 15, 13, 45, 86, 58, 16, 16, 31, 3, 9, 59, 74, 28, 19, 61, 71, 80, 47, 60, 52, 66, 82, 99, 57, 54, 18, 99, 59, 44, 42, 99, 33, 27, 92, 9, 4, 36, 34, 65, 4, 72, 62, 97, 63, 85, 81, 73, 3, 63, 18, 41, 14, 18, 23, 51, 86, 6, 82, 50, 92, 18, 84, 58, 71, 61, 88, 17, 43, 44, 74, 91, 52, 83, 41, 78, 11, 29, 29, 36, 45, 62, 41, 51, 79, 19, 37, 53, 53, 77, 25, 58, 43, 16, 32, 44, 47]

37 numbers were not generated:

[1, 2, 5, 7, 8, 10, 20, 21, 22, 24, 26, 30, 35, 38, 39, 40, 46, 48, 49, 55, 56, 64, 67, 68, 69, 70, 75, 76, 87, 89, 90, 93, 94, 95, 96, 98, 100]

Generated numbers:

[92, 70, 56, 75, 6, 65, 20, 3, 56, 84, 49, 60, 99, 70, 41, 71, 10, 56, 76, 82, 25, 69, 42, 69, 90, 96, 3, 60, 10, 78, 10, 22, 79, 12, 93, 85, 3, 67, 99, 46, 10, 19, 79, 89, 84, 70, 79, 75, 44, 72, 62, 38, 49, 42, 26, 46, 53, 90, 39, 84, 50, 86, 5, 56, 40, 21, 51, 94, 45, 23, 71, 95, 63, 72, 88, 74, 89, 32, 89, 1, 66, 80, 49, 23, 39, 17, 2, 64, 74, 91, 80, 88, 89, 85, 98, 26, 15, 62, 11, 88]

37 numbers were not generated:

[4, 7, 8, 9, 13, 14, 16, 18, 24, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 43, 47, 48, 52, 54, 55, 57, 58, 59, 61, 68, 73, 77, 81, 83, 87, 97, 100]

Generated numbers:

[13, 64, 54, 3, 63, 50, 36, 12, 33, 17, 71, 94, 92, 79, 73, 87, 37, 93, 79, 24, 82, 34, 54, 95, 7, 22, 47, 16, 8, 22, 25, 37, 7, 69, 65, 29, 18, 6, 55, 13, 88, 41, 83, 75, 26, 20, 49, 9, 68, 76, 57, 93, 56, 2, 43, 95, 30, 17, 55, 97, 46, 80, 89, 53, 28, 53, 26, 98, 96, 97, 4, 22, 14, 80, 46, 69, 49, 8, 44, 94, 51, 4, 6, 36, 28, 6, 93, 67, 73, 15, 33, 46, 4, 96, 84, 13, 90, 49, 31, 35]

33 numbers were not generated:

[1, 5, 10, 11, 19, 21, 23, 27, 32, 38, 39, 40, 42, 45, 48, 52, 58, 59, 60, 61, 62, 66, 70, 72, 74, 77, 78, 81, 85, 86, 91, 99, 100]

Generated numbers:

[18, 64, 86, 21, 84, 29, 55, 27, 39, 94, 5, 92, 34, 7, 49, 16, 48, 12, 18, 1, 40, 9, 56, 48, 52, 71, 86, 73, 87, 40, 71, 83, 13, 28, 60, 52, 96, 11, 40, 34, 64, 50, 97, 55, 43, 88, 40, 13, 55, 1, 42, 7, 53, 57, 96, 81, 30, 36, 64, 91, 93, 46, 21, 34, 1, 91, 96, 66, 1, 96,

20, 87, 3, 82, 31, 7, 59, 94, 72, 92, 58, 36, 11, 9, 63, 44, 99, 23, 24, 55, 13, 64, 97, 23, 84, 59, 47, 11, 20, 92]

41 numbers were not generated:

[2, 4, 6, 8, 10, 14, 15, 17, 19, 22, 25, 26, 32, 33, 35, 37, 38, 41, 45, 51, 54, 61, 62, 65, 67, 68, 69, 70, 74, 75, 76, 77, 78, 79, 80, 85, 89, 90, 95, 98, 100]

Generated numbers:

[86, 33, 78, 71, 4, 11, 26, 43, 7, 56, 52, 58, 44, 72, 84, 56, 100, 44, 74, 10, 78, 66, 73, 8, 19, 76, 76, 64, 66, 80, 64, 85, 52, 47, 70, 94, 85, 100, 56, 76, 49, 13, 12, 33, 52, 50, 96, 12, 58, 43, 44, 45, 30, 56, 51, 77, 60, 83, 65, 26, 40, 20, 87, 2, 81, 75, 35, 6, 21, 23, 43, 64, 14, 72, 54, 21, 41, 55, 38, 33, 7, 21, 36, 44, 88, 24, 80, 90, 20, 28, 55, 70, 71, 81, 34, 67, 35, 70, 93, 52]

36 numbers were not generated:

[1, 3, 5, 9, 15, 16, 17, 18, 22, 25, 27, 29, 31, 32, 37, 39, 42, 46, 48, 53, 57, 59, 61, 62, 63, 68, 69, 79, 82, 89, 91, 92, 95, 97, 98, 99]

Generated numbers:

[52, 52, 61, 71, 94, 92, 31, 27, 58, 96, 4, 90, 77, 68, 53, 54, 68, 28, 46, 75, 36, 55, 87, 82, 49, 78, 40, 24, 37, 43, 43, 54, 53, 54, 10, 59, 82, 73, 12, 66, 67, 73, 39, 40, 13, 15, 4, 46, 51, 40, 18, 47, 84, 100, 92, 33, 94, 78, 82, 41, 56, 51, 88, 62, 63, 18, 18, 91, 23, 21, 24, 65, 98, 24, 12, 50, 60, 81, 27, 82, 97, 7, 60, 44, 67, 49, 68, 29, 93, 65, 21, 79, 9, 99, 25, 35, 41, 13, 94, 1]

33 numbers were not generated:

[2, 3, 5, 6, 8, 11, 14, 16, 17, 19, 20, 22, 26, 30, 32, 34, 38, 42, 45, 48, 57, 64, 69, 70, 72, 74, 76, 80, 83, 85, 86, 89, 95]

Generated numbers:

[71, 64, 100, 70, 53, 2, 87, 33, 97, 58, 9, 85, 64, 68, 15, 23, 30, 14, 11, 91, 51, 41, 4, 10, 82, 26, 38, 76, 52, 47, 80, 44, 74, 14, 70, 22, 70, 36, 71, 89, 4, 14, 62, 70, 79, 3, 79, 76, 39, 15, 76, 1, 90, 10, 36, 74, 75, 98, 45, 89, 60, 87, 19, 19, 4, 42, 8, 23, 26, 22, 36, 14, 74, 76, 54, 96, 27, 11, 40, 85, 19, 83, 12, 7, 53, 100, 31, 76, 78, 1, 16, 18, 70, 45, 17, 86, 27, 41, 33, 19]

39 numbers were not generated:

[5, 6, 13, 20, 21, 24, 25, 28, 29, 32, 34, 35, 37, 43, 46, 48, 49, 50, 55, 56, 57, 59, 61, 63, 65, 66, 67, 69, 72, 73, 77, 81, 84, 88, 92, 93, 94, 95, 99]

Generated numbers:

[16, 26, 90, 89, 38, 98, 17, 15, 78, 72, 5, 92, 35, 16, 3, 48, 42, 56, 64, 98, 28, 66, 58, 89, 37, 76, 55, 34, 97, 99, 54, 28, 27, 2, 47, 98, 79, 37, 23, 87, 96, 40, 70, 72, 5, 50, 15, 25, 33, 6, 75, 18, 27, 37, 10, 57, 56, 44, 69, 32, 88, 100, 67, 66, 70, 29, 61, 18, 69,

29, 62, 97, 57, 45, 94, 72, 95, 63, 26, 58, 95, 51, 99, 13, 83, 64, 72, 86, 25, 63, 60, 89, 49, 50, 66, 25, 1, 47, 9, 30]

33 numbers were not generated:

[4, 7, 8, 11, 12, 14, 19, 20, 21, 22, 24, 31, 36, 39, 41, 43, 46, 52, 53, 59, 65, 68, 71, 73, 74, 77, 80, 81, 82, 84, 85, 91, 93]

Generated numbers:

[83, 65, 79, 85, 52, 77, 32, 86, 13, 21, 3, 80, 27, 57, 21, 26, 90, 58, 38, 27, 82, 2, 13, 82, 64, 90, 25, 55, 70, 61, 36, 96, 80, 50, 20, 70, 68, 96, 82, 75, 41, 9, 36, 50, 26, 6, 26, 1, 56, 27, 74, 31, 100, 44, 73, 3, 3, 62, 32, 86, 9, 20, 59, 2, 86, 93, 61, 96, 54, 15,

38, 34, 53, 7, 22, 65, 16, 2, 39, 96, 83, 83, 83, 20, 50, 89, 12, 98, 57, 18, 88, 79, 52, 89, 49, 2, 32, 61, 22, 35]

41 numbers were not generated:

[4, 5, 8, 10, 11, 14, 17, 19, 23, 24, 28, 29, 30, 33, 37, 40, 42, 43, 45, 46, 47, 48, 51, 60, 63, 66, 67, 69, 71, 72, 76, 78, 81, 84, 87, 91, 92, 94, 95, 97, 99]

Generated numbers:

[69, 81, 45, 88, 54, 51, 29, 82, 5, 3, 8, 76, 55, 30, 91, 75, 90, 55, 51, 75, 8, 89, 71, 82, 48, 87, 31, 70, 16, 63, 96, 45, 73, 24, 56, 97, 27, 41, 100, 42, 61, 19, 11, 76, 93, 34, 29, 1, 63, 71, 41, 48, 85, 30, 13, 65, 46, 62, 35, 35, 75, 96, 5, 85, 46, 19, 66, 29, 18, 6, 31, 91, 73, 88, 39, 100, 12, 78, 73, 95, 13, 69, 22, 45, 61, 25, 77, 73, 8, 38, 54, 63, 67, 43, 23, 68, 26, 59, 79, 94]

35 numbers were not generated:

[2, 4, 7, 9, 10, 14, 15, 17, 20, 21, 28, 32, 33, 36, 37, 40, 44, 47, 49, 50, 52, 53, 57, 58, 60, 64, 72, 74, 80, 83, 84, 86, 92, 98, 99]

Generated numbers:

[2, 4, 36, 25, 53, 88, 31, 25, 44, 94, 17, 69, 16, 29, 28, 25, 70, 60, 60, 73, 32, 53, 26, 26, 15, 39, 35, 36, 17, 24, 36, 82, 21, 38, 91, 25, 86, 34, 66, 23, 11, 35, 45, 39, 15, 64, 56, 79, 51, 67, 67, 66, 42, 93, 38, 41, 20, 20, 48, 53, 63, 3, 73, 11, 48, 36, 99, 88, 30, 9, 67, 34, 60, 16, 62, 94, 9, 31, 37, 55, 100, 80, 26, 6, 45, 19, 15, 34, 2, 5, 51, 90, 61, 54, 86, 70, 68, 86, 73, 54]

40 numbers were not generated:

[1, 7, 8, 10, 12, 13, 14, 18, 22, 27, 33, 40, 43, 46, 47, 49, 50, 52, 57, 58, 59, 65, 71, 72, 74, 75, 76, 77, 78, 81, 83, 84, 85, 87, 89, 92, 95, 96, 97, 98]

Generated numbers:

[96, 98, 76, 31, 82, 100, 56, 96, 53, 36, 47, 29, 59, 67, 80, 21, 8, 59, 58, 50, 76, 90, 17, 1, 21, 31, 35, 30, 93, 65, 37, 99, 71, 89, 61, 66, 75, 65, 99, 96, 39, 11, 2, 70, 21, 98, 80, 1, 52, 57, 94, 26, 77, 18, 58, 63, 37, 30, 58, 65, 57, 52, 5, 13, 77, 84, 20, 86, 84,

48, 41, 88, 10, 52, 23, 71, 74, 4, 9, 16, 78, 88, 15, 28, 43, 35, 35, 29, 81, 7, 9, 88, 43, 12, 79, 65, 15, 87, 26, 27]

33 numbers were not generated:

[3, 6, 14, 19, 22, 24, 25, 32, 33, 34, 38, 40, 42, 44, 45, 46, 49, 51, 54, 55, 60, 62, 64, 68, 69, 72, 73, 83, 85, 91, 92, 95, 97]

Generated numbers:

[23, 31, 99, 64, 3, 86, 60, 59, 40, 33, 39, 81, 6, 64, 8, 60, 19, 40, 98, 23, 19, 49, 5, 92, 76, 8, 70, 68, 35, 36, 99, 41, 18, 63, 85, 74, 3, 71, 55, 18, 30, 91, 11, 97, 26, 3, 55, 75, 31, 60, 54, 33, 37, 84, 49, 57, 6, 65, 80, 85, 55, 52, 52, 38, 94, 69, 58, 52, 1, 99, 3, 54, 95, 12, 53, 24, 44, 79, 28, 88, 10, 6, 84, 23, 29, 31, 94, 80, 50, 18, 92, 7, 85, 23, 18, 78, 17, 100, 37, 23]

36 numbers were not generated:

[2, 4, 9, 13, 14, 15, 16, 20, 21, 22, 25, 27, 32, 34, 42, 43, 45, 46, 47, 48, 51, 56, 61, 62, 66, 67, 72, 73, 77, 82, 83, 87, 89, 90, 93, 96]

Generated numbers:

[33, 67, 23, 29, 29, 63, 85, 25, 85, 87, 76, 63, 22, 41, 42, 66, 46, 35, 79, 41, 1, 79, 18, 30, 98, 89, 2, 49, 53, 43, 57, 38, 79, 23, 16, 47, 39, 29, 34, 98, 27, 27, 18, 54, 62, 57, 84, 78, 25, 9, 4, 65, 52, 52, 8, 71, 25, 13, 95, 35, 62, 84, 81, 66, 68, 98, 1, 55, 13, 28, 19, 58, 17, 65, 3, 64, 68, 42, 28, 52, 61, 54, 3, 42, 58, 93, 58, 38, 52, 58, 80, 8, 29, 86, 29, 18, 14, 86, 39, 9]

42 numbers were not generated:

[5, 6, 7, 10, 11, 12, 15, 20, 21, 24, 26, 31, 32, 36, 37, 40, 44, 45, 48, 50, 51, 56, 59, 60, 69, 70, 72, 73, 74, 75, 77, 82, 83, 88, 90, 91, 92, 94, 96, 97, 99, 100]

Generated numbers:

[46, 59, 82, 23, 85, 56, 74, 56, 84, 96, 42, 89, 55, 68, 52, 31, 66, 48, 99, 91, 73, 45, 81, 57, 80, 95, 25, 77, 38, 18, 92, 37, 100, 6, 10, 19, 21, 99, 13, 63, 41, 37, 42, 20, 28, 34, 65, 82, 28, 78, 64, 17, 6, 76, 51, 31, 14, 83, 86, 21, 77, 50, 63, 74, 4, 96, 30, 37, 76, 49, 58, 12, 62, 20, 36, 61, 54, 65, 31, 21, 37, 81, 62, 30, 2, 70, 93, 63, 36, 47, 96, 53, 66, 84, 87, 55, 40, 96, 27, 67]

30 numbers were not generated:

[1, 3, 5, 7, 8, 9, 11, 15, 16, 22, 24, 26, 29, 32, 33, 35, 39, 43, 44, 60, 69, 71, 72, 75, 79, 88, 90, 94, 97, 98]

Generated numbers:

[21, 27, 79, 32, 86, 1, 13, 44, 3, 4, 89, 53, 96, 21, 57, 38, 74, 36, 46, 79, 23, 1, 21, 88, 67, 14, 20, 91, 96, 41, 53, 70, 17, 86, 50, 70, 6, 91, 41, 11, 91, 45, 100, 14, 48, 44, 88, 62, 93, 59, 34, 14, 22, 52, 48, 60, 84, 20, 10, 2, 96, 92, 32, 56, 61, 29, 22, 4, 57, 45, 67, 84, 50, 52, 49, 71, 94, 94, 43, 35, 30, 8, 31, 97, 88, 1, 30, 74, 61, 71, 61, 15, 2, 75, 91, 60, 96, 83, 84, 30]

41 numbers were not generated:

[5, 7, 9, 12, 16, 18, 19, 24, 25, 26, 28, 33, 37, 39, 40, 42, 47, 51, 54, 55, 58, 63, 64, 65, 66, 68, 69, 72, 73, 76, 77, 78, 80, 81, 82, 85, 87, 90, 95, 98, 99]

Generated numbers:

[35, 2, 15, 24, 42, 12, 98, 38, 56, 40, 41, 9, 10, 90, 69, 4, 14, 59, 97, 64, 10, 23, 43, 25, 91, 43, 20, 1, 38, 20, 79, 50, 4, 46, 72, 14, 15, 14, 20, 16, 56, 26, 91, 21, 18, 71, 92, 89, 4, 100, 70, 71, 84, 24, 42, 14, 27, 71, 92, 89, 92, 51, 100, 12, 60, 61, 12, 4, 2, 86, 48, 37, 97, 41, 71, 83, 83, 14, 16, 44, 10, 62, 32, 59, 83, 22, 76, 37, 48, 64, 53, 36, 95, 58, 84, 100, 54, 87, 23, 42]

41 numbers were not generated:

[3, 5, 6, 7, 8, 11, 13, 17, 19, 28, 29, 30, 31, 33, 34, 39, 45, 47, 49, 52, 55, 57, 63, 65, 66, 67, 68, 73, 74, 75, 77, 78, 80, 81, 82, 85, 88, 93, 94, 96, 99]

Generated numbers:

[30, 81, 55, 2, 31, 17, 20, 73, 51, 25, 74, 28, 82, 16, 70, 18, 72, 2, 49, 62, 91, 70, 70, 34, 65, 49, 86, 22, 6, 71, 47, 40, 56, 86, 1, 13, 59, 54, 60, 45, 94, 54, 11, 98, 93, 29, 10, 83, 17, 38, 92, 16, 86, 54, 94, 16, 39, 15, 4, 84, 83, 47, 23, 22, 10, 92, 32, 51, 54, 46, 93, 38, 63, 1, 44, 85, 57, 58, 52, 88, 98, 11, 73, 9, 15, 28, 62, 98, 10, 16, 59, 36, 31, 91, 27, 54, 51, 97, 8, 64]

36 numbers were not generated:

[3, 5, 7, 12, 14, 19, 21, 24, 26, 33, 35, 37, 41, 42, 43, 48, 50, 53, 61, 66, 67, 68, 69, 75, 76, 77, 78, 79, 80, 87, 89, 90, 95, 96, 99, 100]

Generated numbers:

[42, 69, 87, 92, 53, 95, 24, 69, 27, 83, 71, 60, 20, 17, 60, 78, 58, 32, 17, 22, 25, 50, 49, 89, 5, 99, 61, 33, 80, 85, 40, 58, 22, 13, 44, 36, 64, 95, 60, 27, 87, 5, 68, 8, 16, 42, 14, 73, 94, 91, 89, 95, 22, 62, 42, 94, 60, 84, 18, 39, 10, 64, 63, 51, 49, 53, 27, 91, 65, 57, 86, 39, 90, 87, 90, 72, 93, 27, 69, 94, 24, 51, 8, 27, 88, 58, 100, 89, 12, 17, 54, 31, 92, 40, 42, 89, 68, 82, 69, 70]

41 numbers were not generated:

[1, 2, 3, 4, 6, 7, 9, 11, 15, 19, 21, 23, 26, 28, 29, 30, 34, 35, 37, 38, 41, 43, 45, 46, 47, 48, 52, 55, 56, 59, 66, 67, 74, 75, 76, 77, 79, 81, 96, 97, 98]

Generated numbers:

[53, 8, 19, 40, 11, 30, 48, 92, 85, 25, 73, 12, 38, 41, 52, 70, 77, 21, 67, 17, 75, 67, 96, 84, 10, 11, 58, 80, 3, 66, 78, 55, 54, 37, 54, 78, 73, 1, 33, 68, 53, 45, 78, 71, 55, 67, 22, 80, 5, 93, 5, 65, 4, 22, 93, 2, 94, 41, 2, 62, 61, 34, 9, 77, 33, 15, 77, 13, 19, 65, 90, 29, 22, 96, 10, 30, 40, 84, 88, 31, 8, 81, 53, 98, 88, 65, 91, 32, 36, 9, 75, 89, 64, 70, 64, 60, 11, 34, 98, 48]

38 numbers were not generated:

[6, 7, 14, 16, 18, 20, 23, 24, 26, 27, 28, 35, 39, 42, 43, 44, 46, 47, 49, 50, 51, 56, 57, 59, 63, 69, 72, 74, 76, 79, 82, 83, 86, 87, 95, 97, 99, 100]

Generated numbers:

[87, 49, 22, 12, 12, 4, 18, 78, 55, 86, 7, 83, 84, 19, 4, 94, 86, 63, 35, 25, 43, 1, 82, 83, 24, 97, 90, 32, 56, 25, 32, 21, 73, 1, 10, 4, 52, 57, 57, 47, 53, 70, 44, 38, 78, 17, 77, 32, 6, 73, 79, 58, 41, 58, 6, 80, 41, 55, 59, 26, 95, 23, 84, 22, 20, 30, 59, 21, 10, 33,

93, 14, 41, 63, 44, 33, 68, 74, 88, 82, 20, 70, 65, 48, 9, 68, 34, 99, 6, 28, 16, 48, 59, 53, 49, 52, 70, 59, 35, 67]

38 numbers were not generated:

[2, 3, 5, 8, 11, 13, 15, 27, 29, 31, 36, 37, 39, 40, 42, 45, 46, 50, 51, 54, 60, 61, 62, 64, 66, 69, 71, 72, 75, 76, 81, 85, 89, 91, 92, 96, 98, 100]

Generated numbers:

[10, 94, 95, 99, 98, 8, 36, 33, 66, 14, 13, 15, 56, 63, 37, 92, 18, 94, 80, 60, 77, 88, 25, 15, 68, 84, 93, 34, 72, 29, 18, 53, 72, 45, 21, 73, 7, 11, 49, 79, 10, 24, 36, 68, 84, 42, 27, 37, 46, 34, 89, 66, 7, 71, 83, 14, 96, 75, 30, 97, 19, 94, 35, 6, 56, 24, 94, 64, 8, 99, 24, 46, 26, 32, 34, 71, 3, 64, 91, 100, 92, 57, 70, 63, 51, 77, 47, 40, 6, 71, 75, 43, 18, 43, 11, 29, 13, 16, 38, 65]

35 numbers were not generated:

[1, 2, 4, 5, 9, 12, 17, 20, 22, 23, 28, 31, 39, 41, 44, 48, 50, 52, 54, 55, 58, 59, 61, 62, 67, 69, 74, 76, 78, 81, 82, 85, 86, 87, 90]

Generated numbers:

[49, 14, 90, 41, 75, 10, 22, 27, 75, 63, 55, 88, 62, 83, 85, 9, 81, 3, 29, 50, 69, 74, 60, 4, 50, 66, 7, 70, 74, 66, 32, 33, 70, 25, 44, 93, 35, 58, 85, 82, 91, 84, 24, 13, 27, 47, 23, 57, 87, 3, 45, 91, 7, 3, 34, 27, 81, 30, 58, 6, 6, 44, 45, 62, 88, 65, 1, 11, 44, 86, 12, 53, 92, 68, 90, 15, 77, 2, 96, 75, 64, 57, 9, 24, 94, 80, 75, 18, 66, 45, 59, 9, 35, 56, 67, 57, 64, 4, 58, 79]

34 numbers were not generated:

[5, 8, 16, 17, 19, 20, 21, 26, 28, 31, 36, 37, 38, 39, 40, 42, 43, 46, 48, 51, 52, 54, 61, 71, 72, 73, 76, 78, 89, 95, 97, 98, 99, 100]

Generated numbers:

[35, 69, 58, 79, 20, 65, 35, 60, 25, 81, 69, 68, 1, 5, 85, 65, 24, 36, 37, 12, 58, 54, 68, 88, 42, 50, 68, 28, 47, 75, 71, 79, 61, 10, 14, 11, 9, 37, 99, 44, 5, 35, 6, 76, 67, 79, 95, 19, 50, 55, 46, 90, 12, 78, 5, 70, 96, 76, 51, 89, 16, 49, 94, 96, 75, 15, 14, 25, 18, 98, 18, 98, 93, 31, 54, 97, 3, 47, 14, 70, 19, 39, 52, 18, 20, 23, 16, 5, 52, 16, 13, 11, 17, 83, 99, 31, 29, 2, 1, 16]

37 numbers were not generated:

[4, 7, 8, 21, 22, 26, 27, 30, 32, 33, 34, 38, 40, 41, 43, 45, 48, 53, 56, 57, 59, 62, 63, 64, 66, 72, 73, 74, 77, 80, 82, 84, 86, 87, 91, 92, 100]

Generated numbers:

[17, 16, 46, 12, 48, 54, 48, 56, 56, 29, 99, 81, 58, 92, 23, 65, 58, 33, 10, 76, 27, 34, 86, 94, 81, 33, 71, 97, 35, 93, 48, 59, 67, 69, 3, 50, 6, 49, 63, 20, 46, 67, 82, 19, 98, 83, 38, 21, 27, 98, 86, 25, 94, 30, 21, 43, 13, 75, 36, 9, 57, 17, 19, 87, 51, 84, 56, 84, 68, 91, 3, 19, 24, 95, 37, 40, 64, 6, 23, 91, 70, 53, 75, 68, 66, 19, 54, 68, 4, 65, 76, 31, 40, 100, 84, 23, 36, 15, 44, 2]

33 numbers were not generated:

[1, 5, 7, 8, 11, 14, 18, 22, 26, 28, 32, 39, 41, 42, 45, 47, 52, 55, 60, 61, 62, 72, 73, 74, 77, 78, 79, 80, 85, 88, 89, 90, 96]

Generated numbers:

[42, 65, 88, 62, 39, 25, 100, 57, 10, 87, 6, 100, 71, 48, 79, 90, 92, 89, 18, 49, 81, 69, 7, 29, 100, 18, 46, 6, 85, 39, 96, 44, 23, 16, 71, 84, 20, 94, 22, 23, 48, 66, 46, 32, 52, 40, 34, 95, 50, 58, 90, 34, 47, 49, 22, 65, 82, 76, 87, 15, 17, 53, 76, 52, 79, 5, 51, 84, 21, 55, 91, 27, 61, 25, 40, 5, 42, 43, 24, 61, 68, 32, 10, 75, 38, 81, 64, 26, 21, 48, 87, 99, 60, 61, 45, 42, 58, 55, 4, 57]

35 numbers were not generated:

[1, 2, 3, 8, 9, 11, 12, 13, 14, 19, 28, 30, 31, 33, 35, 36, 37, 41, 54, 56, 59, 63, 67, 70, 72, 73, 74, 77, 78, 80, 83, 86, 93, 97, 98]

Generated numbers:

[84, 17, 83, 12, 21, 17, 20, 21, 63, 86, 94, 3, 31, 72, 76, 73, 33, 23, 95, 41, 94, 68, 64, 32, 24, 84, 10, 42, 97, 39, 68, 25, 99, 80, 19, 10, 64, 80, 90, 41, 3, 44, 12, 60, 98, 32, 34, 69, 71, 45, 24, 98, 82, 81, 48, 29, 69, 56, 86, 13, 21, 73, 91, 87, 38, 18, 18, 2, 42, 76, 36, 8, 79, 3, 23, 71, 11, 19, 41, 48, 98, 100, 41, 77, 66, 59, 47, 71, 7, 56, 89, 100, 93, 39, 77, 87, 5, 99, 28, 61]

36 numbers were not generated:

[1, 4, 6, 9, 14, 15, 16, 22, 26, 27, 30, 35, 37, 40, 43, 46, 49, 50, 51, 52, 53, 54, 55, 57, 58, 62, 65, 67, 70, 74, 75, 78, 85, 88, 92, 96]

Generated numbers:

[39, 42, 64, 93, 73, 21, 95, 56, 46, 24, 67, 92, 26, 55, 63, 12, 57, 42, 49, 33, 63, 2, 31, 37, 96, 75, 59, 79, 86, 92, 88, 14, 96, 14, 76, 8, 63, 12, 1, 62, 30, 69, 79, 80, 37, 75, 60, 24, 44, 99, 90, 63, 71, 54, 90, 99, 93, 70, 65, 54, 86, 97, 35, 81, 81, 74, 3, 10, 51,

39, 58, 30, 61, 14, 52, 75, 78, 41, 53, 27, 34, 60, 91, 7, 13, 90, 17, 57, 32, 100, 39, 42, 99, 40, 17, 86, 70, 83, 56, 97]

33 numbers were not generated:

[4, 5, 6, 9, 11, 15, 16, 18, 19, 20, 22, 23, 25, 28, 29, 36, 38, 43, 45, 47, 48, 50, 66, 68, 72, 77, 82, 84, 85, 87, 89, 94, 98]

Generated numbers:

[24, 31, 10, 47, 84, 23, 100, 62, 6, 19, 12, 19, 29, 54, 75, 63, 55, 89, 16, 33, 94, 53, 58, 28, 78, 66, 46, 86, 96, 97, 72, 34, 5, 80, 95, 18, 19, 2, 20, 35, 70, 40, 75, 100, 28, 35, 26, 33, 38, 28, 36, 37, 87, 70, 60, 7, 60, 14, 5, 61, 6, 91, 79, 74, 81, 16, 68, 86, 68,

97, 56, 90, 47, 29, 10, 71, 72, 58, 74, 49, 54, 75, 1, 43, 59, 3, 74, 37, 48, 9, 31, 81, 44, 35, 58, 73, 18, 88, 31, 46]

33 numbers were not generated:

[4, 8, 11, 13, 15, 17, 21, 22, 25, 27, 30, 32, 39, 41, 42, 45, 50, 51, 52, 57, 64, 65, 67, 69, 76, 77, 82, 83, 85, 92, 93, 98, 99]

Generated numbers:

[35, 91, 35, 13, 93, 65, 85, 93, 25, 9, 99, 4, 14, 64, 82, 54, 71, 44, 9, 12, 76, 73, 14, 92, 8, 7, 57, 53, 13, 88, 30, 25, 30, 36, 49, 2, 99, 3, 18, 29, 33, 60, 40, 31, 71, 82, 18, 13, 33, 91, 42, 48, 18, 63, 13, 33, 69, 85, 99, 92, 14, 28, 60, 100, 78, 86, 34, 71, 44, 82, 13, 45, 50, 93, 76, 69, 32, 69, 35, 33, 20, 19, 10, 63, 52, 71, 49, 17, 29, 13, 52, 93, 65, 1, 42, 70, 22, 71, 71, 33]

44 numbers were not generated:

[5, 6, 11, 15, 16, 21, 23, 24, 26, 27, 37, 38, 39, 41, 43, 46, 47, 51, 55, 56, 58, 59, 61, 62, 66, 67, 68, 72, 74, 75, 77, 79, 80, 81, 83, 84, 87, 89, 90, 94, 95, 96, 97, 98]

Generated numbers:

[22, 2, 78, 7, 52, 43, 14, 61, 48, 62, 60, 78, 62, 81, 86, 65, 87, 94, 33, 11, 95, 17, 54, 91, 39, 39, 38, 7, 55, 79, 13, 21, 4, 66, 20, 16, 10, 50, 28, 26, 98, 4, 96, 7, 8, 10, 47, 72, 19, 8, 46, 65, 87, 99, 7, 40, 89, 44, 30, 4, 49, 21, 87, 73, 96, 70, 79, 27, 74, 8, 67, 22, 75, 39, 36, 96, 89, 69, 66, 95, 28, 9, 91, 49, 87, 89, 24, 21, 65, 68, 70, 35, 40, 77, 81, 61, 32, 69, 82, 83]

35 numbers were not generated:

[1, 3, 5, 6, 12, 15, 18, 23, 25, 29, 31, 34, 37, 41, 42, 45, 51, 53, 56, 57, 58, 59, 63, 64, 71, 76, 80, 84, 85, 88, 90, 92, 93, 97, 100]

Generated numbers:

[52, 73, 25, 17, 82, 17, 42, 100, 69, 65, 77, 72, 43, 38, 75, 64, 65, 42, 4, 10, 60, 97, 66, 19, 94, 76, 71, 86, 70, 15, 89, 15, 98, 53, 98, 37, 59, 32, 35, 86, 25, 47, 57, 59, 2, 50, 57, 34, 62, 83, 73, 11, 68, 84, 78, 20, 77, 44, 68, 91, 32, 39, 61, 35, 62, 50, 25, 40, 50, 40, 57, 9, 47, 64, 61, 72, 9, 35, 56, 73, 33, 46, 29, 99, 87, 12, 84, 27, 98, 99, 90, 79, 33, 36, 25, 45, 28, 38, 21, 73]

34 numbers were not generated:

[1, 3, 5, 6, 7, 8, 13, 14, 16, 18, 22, 23, 24, 26, 30, 31, 41, 48, 49, 51, 54, 55, 58, 63, 67, 74, 80, 81, 85, 88, 92, 93, 95, 96]

Generated numbers:

[54, 61, 91, 42, 9, 23, 84, 23, 23, 18, 16, 81, 55, 2, 95, 38, 56, 23, 22, 56, 14, 70, 75, 93, 29, 65, 14, 74, 58, 28, 25, 19, 44, 54, 32, 59, 58, 22, 19, 35, 12, 31, 76, 60, 83, 92, 67, 55, 27, 100, 71, 28, 38, 16, 64, 26, 24, 10, 60, 35, 24, 33, 43, 22, 33, 100, 62, 20,

8, 46, 5, 76, 33, 10, 2, 81, 62, 52, 26, 21, 67, 83, 40, 87, 28, 85, 16, 93, 21, 99, 92, 78, 45, 33, 21, 90, 41, 4, 53, 56]

36 numbers were not generated:

[1, 3, 6, 7, 11, 13, 15, 17, 30, 34, 36, 37, 39, 47, 48, 49, 50, 51, 57, 63, 66, 68, 69, 72, 73, 77, 79, 80, 82, 86, 88, 89, 94, 96, 97, 98]

Generated numbers:

[32, 54, 87, 91, 29, 76, 51, 21, 78, 78, 29, 96, 94, 79, 68, 51, 9, 57, 77, 75, 82, 17, 91, 65, 4, 84, 73, 70, 63, 53, 36, 12, 23, 21, 59, 23, 56, 87, 3, 19, 13, 97, 82, 20, 69, 8, 11, 4, 66, 67, 77, 28, 52, 79, 98, 12, 4, 85, 47, 28, 93, 7, 59, 38, 17, 32, 64, 51, 62, 26, 99, 53, 68, 73, 21, 71, 36, 92, 71, 6, 82, 28, 22, 84, 92, 75, 66, 97, 84, 98, 13, 14, 69, 83, 75, 96, 37, 94, 30, 13]

39 numbers were not generated:

[1, 2, 5, 10, 15, 16, 18, 24, 25, 27, 31, 33, 34, 35, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 55, 58, 60, 61, 72, 74, 80, 81, 86, 88, 89, 90, 95, 100]

Generated numbers:

[45, 25, 17, 57, 8, 87, 1, 48, 23, 24, 77, 34, 55, 58, 43, 89, 37, 71, 100, 99, 84, 29, 82, 48, 76, 41, 8, 33, 30, 72, 44, 17, 45, 74, 85, 90, 29, 28, 92, 82, 72, 37, 11, 73, 64, 14, 42, 13, 3, 78, 27, 58, 86, 3, 80, 96, 81, 94, 29, 48, 37, 46, 70, 59, 68, 17, 94, 42, 73,

74, 71, 11, 47, 45, 6, 39, 20, 55, 14, 87, 2, 83, 4, 77, 72, 29, 3, 14, 93, 6, 13, 43, 48, 100, 17, 62, 1, 83, 9, 53]

37 numbers were not generated:

[5, 7, 10, 12, 15, 16, 18, 19, 21, 22, 26, 31, 32, 35, 36, 38, 40, 49, 50, 51, 52, 54, 56, 60, 61, 63, 65, 66, 67, 69, 75, 79, 88, 91, 95, 97, 98]

Generated numbers:

[73, 75, 28, 90, 42, 83, 30, 91, 94, 22, 41, 13, 24, 74, 74, 83, 2, 39, 60, 52, 1, 27, 76, 31, 100, 9, 68, 43, 47, 34, 44, 15, 64, 45, 54, 16, 41, 41, 45, 95, 84, 38, 19, 96, 31, 4, 73, 5, 87, 24, 63, 75, 82, 69, 76, 81, 56, 54, 32, 65, 37, 75, 98, 35, 100, 50, 52, 4, 92,

43, 88, 90, 20, 80, 32, 67, 39, 21, 100, 80, 69, 91, 38, 80, 50, 64, 38, 70, 44, 72, 59, 88, 64, 83, 96, 72, 95, 62, 72, 6]

36 numbers were not generated:

[3, 7, 8, 10, 11, 12, 14, 17, 18, 23, 25, 26, 29, 33, 36, 40, 46, 48, 49, 51, 53, 55, 57, 58, 61, 66, 71, 77, 78, 79, 85, 86, 89, 93, 97, 99]

Generated numbers:

[11, 59, 41, 99, 64, 8, 45, 57, 65, 47, 94, 58, 47, 71, 44, 10, 94, 41, 70, 46, 47, 68, 61, 22, 60, 9, 75, 23, 66, 80, 42, 81, 56, 3, 5, 35, 22, 76, 61, 44, 93, 58, 46, 21, 35, 83, 3, 1, 42, 9, 81, 50, 61, 49, 61, 99, 11, 71, 13, 23, 69, 11, 54, 5, 10, 83, 74, 33, 53, 39,

51, 60, 38, 85, 9, 94, 8, 16, 54, 93, 26, 23, 96, 67, 43, 22, 71, 62, 86, 34, 84, 11, 58, 78, 5, 33, 82, 64, 9, 43]

40 numbers were not generated:

[2, 4, 6, 7, 12, 14, 15, 17, 18, 19, 20, 24, 25, 27, 28, 29, 30, 31, 32, 36, 37, 40, 48, 52, 55, 63, 72, 73, 77, 79, 87, 88, 89, 90, 91, 92, 95, 97, 98, 100]

Generated numbers:

[62, 94, 19, 88, 43, 25, 13, 75, 87, 69, 30, 64, 23, 56, 11, 69, 98, 30, 20, 28, 98, 18, 24, 58, 84, 22, 77, 27, 77, 79, 55, 86, 10, 95, 3, 7, 98, 45, 59, 87, 5, 58, 77, 98, 8, 78, 100, 44, 53, 15, 86, 6, 52, 48, 94, 9, 65, 100, 20, 30, 4, 12, 8, 33, 47, 81, 76, 20, 100, 80, 13, 13, 81, 57, 35, 25, 61, 57, 60, 37, 18, 51, 95, 74, 34, 13, 19, 35, 64, 16, 75, 74, 26, 97, 84, 64, 65, 1, 82, 6]

34 numbers were not generated:

[2, 14, 17, 21, 29, 31, 32, 36, 38, 39, 40, 41, 42, 46, 49, 50, 54, 63, 66, 67, 68, 70, 71, 72, 73, 83, 85, 89, 90, 91, 92, 93, 96, 99]

Generated numbers:

[84, 25, 32, 9, 81, 68, 47, 58, 27, 32, 48, 34, 83, 47, 17, 100, 70, 65, 40, 38, 32, 74, 60, 51, 52, 7, 45, 82, 35, 44, 9, 80, 42, 18, 14, 97, 73, 50, 23, 66, 23, 24, 20, 86, 52, 30, 71, 81, 51, 27, 48, 32, 15, 21, 22, 44, 57, 18, 29, 77, 91, 42, 89, 77, 49, 53, 38, 15, 15, 82, 83, 87, 100, 46, 75, 37, 79, 81, 7, 80, 54, 52, 91, 46, 29, 57, 48, 58, 68, 30, 1, 27, 4, 76, 37, 96, 54, 96, 41, 6]

37 numbers were not generated:

[2, 3, 5, 8, 10, 11, 12, 13, 16, 19, 26, 28, 31, 33, 36, 39, 43, 55, 56, 59, 61, 62, 63, 64, 67, 69, 72, 78, 85, 88, 90, 92, 93, 94, 95, 98, 99]

Generated numbers:

[46, 50, 70, 73, 38, 24, 66, 69, 50, 78, 90, 41, 18, 3, 14, 74, 2, 21, 42, 79, 36, 24, 18, 17, 74, 77, 33, 50, 85, 13, 99, 67, 96, 49, 47, 79, 91, 26, 82, 89, 58, 82, 68, 61, 37, 96, 78, 77, 47, 97, 73, 17, 61, 92, 1, 6, 82, 98, 31, 90, 98, 21, 74, 40, 62, 3, 25, 32, 92, 92, 18, 81, 8, 6, 68, 4, 15, 27, 35, 58, 56, 65, 97, 26, 53, 24, 74, 76, 60, 87, 79, 82, 36, 29, 5, 32, 84, 35, 84, 1]

37 numbers were not generated:

[7, 9, 10, 11, 12, 16, 19, 20, 22, 23, 28, 30, 34, 39, 43, 44, 45, 48, 51, 52, 54, 55, 57, 59, 63, 64, 71, 72, 75, 80, 83, 86, 88, 93, 94, 95, 100]

Generated numbers:

[31, 19, 93, 86, 74, 97, 73, 65, 34, 56, 11, 75, 51, 78, 48, 91, 99, 20, 35, 43, 3, 63, 66, 2, 23, 49, 79, 64, 93, 79, 93, 82, 57, 87, 66, 51, 21, 29, 59, 65, 38, 37, 34, 60, 34, 57, 27, 97, 29, 33, 50, 9, 39, 64, 45, 80, 23, 91, 71, 36, 28, 67, 96, 68, 45, 28, 80, 65, 46, 56, 47, 17, 45, 28, 87, 1, 4, 83, 61, 36, 75, 45, 88, 93, 45, 77, 22, 11, 5, 81, 30, 30, 20, 7, 34, 94, 43, 92, 38, 34]

34 numbers were not generated:

[6, 8, 10, 12, 13, 14, 15, 16, 18, 24, 25, 26, 32, 40, 41, 42, 44, 52, 53, 54, 55, 58, 62, 69, 70, 72, 76, 84, 85, 89, 90, 95, 98, 100]

Generated numbers:

[60, 36, 55, 54, 64, 22, 15, 74, 23, 2, 59, 98, 11, 71, 95, 84, 98, 31, 35, 21, 59, 55, 12, 12, 100, 67, 3, 45, 50, 56, 84, 65, 68, 28, 51, 42, 84, 96, 74, 24, 43, 45, 94, 75, 97, 2, 58, 47, 75, 84, 6, 74, 83, 91, 56, 1, 4, 14, 63, 53, 89, 2, 11, 46, 69, 70, 25, 100, 5, 12, 9, 48, 96, 27, 55, 54, 84, 20, 3, 32, 40, 89, 29, 7, 56, 62, 16, 5, 92, 13, 83, 90, 68, 47, 26, 76, 81, 8, 72, 47]

29 numbers were not generated:

[10, 17, 18, 19, 30, 33, 34, 37, 38, 39, 41, 44, 49, 52, 57, 61, 66, 73, 77, 78, 79, 80, 82, 85, 86, 87, 88, 93, 99]

Generated numbers:

[43, 12, 4, 1, 70, 27, 7, 71, 47, 5, 4, 8, 75, 67, 100, 33, 17, 80, 72, 86, 39, 66, 86, 53, 62, 85, 22, 2, 35, 54, 50, 86, 9, 38, 87, 38, 1, 19, 39, 26, 81, 67, 32, 50, 5, 62, 23, 7, 69, 22, 84, 65, 100, 46, 73, 15, 50, 32, 30, 3, 6, 16, 85, 55, 95, 78, 87, 71, 31, 90, 46, 80, 4, 58, 68, 44, 34, 82, 2, 35, 14, 23, 40, 8, 95, 24, 53, 14, 12, 23, 40, 76, 99, 76, 85, 67, 10, 94, 34, 31]

36 numbers were not generated:

[11, 13, 18, 20, 21, 25, 28, 29, 36, 37, 41, 42, 45, 48, 49, 51, 52, 56, 57, 59, 60, 61, 63, 64, 74, 77, 79, 83, 88, 89, 91, 92, 93, 96, 97, 98]

Generated numbers:

[72, 60, 33, 90, 84, 54, 37, 6, 81, 92, 75, 38, 62, 51, 93, 9, 79, 99, 71, 36, 27, 55, 81, 39, 10, 41, 73, 74, 24, 32, 70, 17, 4, 85, 86, 76, 43, 28, 74, 16, 31, 90, 31, 34, 59, 23, 76, 35, 76, 3, 31, 51, 73, 76, 77, 89, 99, 43, 23, 23, 45, 78, 69, 39, 35, 58, 63, 69, 16,

75, 54, 17, 43, 80, 11, 93, 82, 73, 25, 73, 44, 55, 54, 79, 59, 77, 49, 29, 78, 58, 36, 32, 12, 5, 27, 4, 5, 90, 100, 36]

39 numbers were not generated:

[1, 2, 7, 8, 13, 14, 15, 18, 19, 20, 21, 22, 26, 30, 40, 42, 46, 47, 48, 50, 52, 53, 56, 57, 61, 64, 65, 66, 67, 68, 83, 87, 88, 91, 94, 95, 96, 97, 98]

Generated numbers:

[74, 28, 23, 71, 10, 37, 48, 90, 44, 68, 48, 53, 41, 39, 14, 37, 56, 85, 55, 87, 13, 97, 94, 79, 53, 78, 61, 76, 74, 10, 40, 81, 3, 96, 61, 18, 10, 7, 34, 65, 8, 20, 71, 75, 13, 74, 65, 67, 85, 49, 22, 10, 97, 4, 64, 73, 9, 29, 1, 57, 68, 87, 71, 42, 76, 49, 98, 76, 67, 34, 66, 26, 29, 85, 53, 100, 53, 51, 59, 44, 47, 91, 13, 55, 90, 75, 98, 34, 9, 31, 10, 15, 49, 63, 49, 49, 61, 1, 43, 74]

41 numbers were not generated:

[2, 5, 6, 11, 12, 16, 17, 19, 21, 24, 25, 27, 30, 32, 33, 35, 36, 38, 45, 46, 50, 52, 54, 58, 60, 62, 69, 70, 72, 77, 80, 82, 83, 84, 86, 88, 89, 92, 93, 95, 99]

Generated numbers:

[43, 97, 48, 81, 70, 63, 16, 81, 84, 82, 95, 90, 50, 75, 5, 64, 27, 32, 21, 29, 22, 67, 27, 1, 81, 15, 17, 100, 81, 36, 78, 75, 23, 40, 69, 20, 76, 56, 77, 56, 5, 46, 90, 36, 17, 93, 30, 33, 47, 17, 19, 48, 84, 10, 64, 84, 9, 3, 93, 78, 36, 42, 41, 23, 13, 62, 44, 44, 12,

80, 44, 31, 2, 74, 70, 96, 59, 51, 80, 67, 56, 55, 82, 98, 55, 68, 41, 3, 47, 44, 14, 76, 79, 3, 27, 21, 41, 25, 19, 84]

38 numbers were not generated:

[4, 6, 7, 8, 11, 18, 24, 26, 28, 34, 35, 37, 38, 39, 45, 49, 52, 53, 54, 57, 58, 60, 61, 65, 66, 71, 72, 73, 83, 85, 86, 87, 88, 89, 91, 92, 94, 99]

Generated numbers:

[72, 23, 93, 81, 4, 63, 89, 43, 50, 61, 59, 29, 60, 37, 62, 19, 93, 48, 92, 90, 82, 41, 65, 84, 80, 20, 52, 56, 86, 41, 75, 23, 17, 7, 88, 3, 91, 9, 82, 17, 59, 70, 79, 72, 94, 53, 33, 54, 99, 57, 18, 48, 3, 37, 72, 8, 70, 37, 15, 67, 63, 78, 31, 16, 54, 7, 36, 70, 25, 73, 50, 21, 49, 32, 80, 18, 69, 80, 96, 39, 20, 64, 88, 68, 85, 54, 35, 59, 33, 69, 4, 2, 42, 66, 86, 51, 70, 46, 71, 10]

30 numbers were not generated:

[1, 5, 6, 11, 12, 13, 14, 22, 24, 26, 27, 28, 30, 34, 38, 40, 44, 45, 47, 55, 58, 74, 76, 77, 83, 87, 95, 97, 98, 100]

Generated numbers:

[41, 5, 100, 11, 16, 39, 68, 51, 93, 89, 85, 64, 39, 43, 90, 82, 75, 7, 86, 4, 10, 71, 53, 44, 25, 74, 11, 24, 17, 55, 28, 69, 73, 2, 72, 16, 92, 72, 96, 49, 51, 13, 82, 33, 96, 91, 3, 28, 24, 57, 55, 2, 83, 36, 9, 68, 46, 51, 9, 47, 94, 82, 37, 57, 34, 33, 63, 20, 46, 76, 93, 60, 97, 9, 12, 47, 38, 21, 71, 33, 93, 88, 87, 40, 72, 33, 20, 39, 19, 88, 1, 55, 15, 83, 70, 23, 77, 43, 83, 96]

34 numbers were not generated:

[6, 8, 14, 18, 22, 26, 27, 29, 30, 31, 32, 35, 42, 45, 48, 50, 52, 54, 56, 58, 59, 61, 62, 65, 66, 67, 78, 79, 80, 81, 84, 95, 98, 99]

Generated numbers:

[17, 24, 59, 30, 1, 49, 95, 50, 46, 90, 15, 1, 55, 17, 76, 17, 85, 28, 25, 20, 45, 50, 32, 66, 19, 86, 19, 65, 33, 77, 12, 24, 86, 50, 85, 45, 85, 40, 64, 53, 7, 94, 5, 77, 9, 91, 99, 16, 100, 65, 26, 96, 30, 76, 77, 41, 85, 1, 80, 34, 71, 3, 60, 72, 90, 1, 35, 63, 73, 75, 94, 41, 36, 26, 89, 38, 80, 8, 54, 59, 86, 81, 60, 40, 35, 51, 98, 71, 17, 45, 59, 15, 53, 84, 94, 80, 21, 20, 55, 24]

40 numbers were not generated:

[2, 4, 6, 10, 11, 13, 14, 18, 22, 23, 27, 29, 31, 37, 39, 42, 43, 44, 47, 48, 52, 56, 57, 58, 61, 62, 67, 68, 69, 70, 74, 78, 79, 82, 83, 87, 88, 92, 93, 97]

Generated numbers:

[47, 45, 94, 37, 49, 22, 49, 45, 56, 40, 31, 98, 100, 82, 59, 54, 58, 62, 9, 100, 73, 67, 22, 24, 39, 41, 94, 71, 18, 43, 73, 54, 56, 97, 38, 31, 13, 2, 47, 28, 15, 45, 60, 18, 5, 9, 90, 20, 26, 91, 2, 4, 7, 24, 33, 4, 6, 59, 46, 50, 56, 37, 45, 19, 41, 28, 54, 9, 100, 79, 63, 18, 91, 59, 31, 89, 45, 91, 20, 48, 71, 99, 29, 44, 25, 4, 80, 84, 88, 76, 77, 59, 93, 47, 48, 30, 19, 13, 36, 68]

39 numbers were not generated:

[1, 3, 8, 10, 11, 12, 14, 16, 17, 21, 23, 27, 32, 34, 35, 42, 51, 52, 53, 55, 57, 61, 64, 65, 66, 69, 70, 72, 74, 75, 78, 81, 83, 85, 86, 87, 92, 95, 96]

Generated numbers:

[78, 62, 86, 8, 31, 59, 25, 8, 64, 55, 43, 21, 91, 47, 51, 89, 97, 95, 61, 78, 86, 42, 89, 75, 34, 3, 72, 56, 33, 41, 38, 50, 1, 80, 19, 6, 6, 77, 43, 45, 20, 86, 7, 11, 50, 4, 37, 31, 28, 6, 92, 45, 78, 61, 44, 57, 93, 70, 60, 50, 81, 91, 91, 54, 32, 6, 26, 41, 67, 62, 83, 91, 80, 77, 53, 83, 15, 90, 37, 5, 64, 48, 19, 90, 91, 22, 5, 98, 65, 8, 46, 97, 50, 66, 55, 48, 50, 16, 82, 85]

35 numbers were not generated:

[2, 9, 10, 12, 13, 14, 17, 18, 23, 24, 27, 29, 30, 35, 36, 39, 40, 49, 52, 58, 63, 68, 69, 71, 73, 74, 76, 79, 84, 87, 88, 94, 96, 99, 100]

Generated numbers:

[53, 65, 50, 62, 3, 21, 62, 11, 2, 67, 11, 39, 82, 70, 96, 38, 86, 58, 72, 38, 43, 41, 87, 52, 7, 8, 53, 91, 54, 28, 17, 40, 60, 25, 73, 89, 2, 79, 38, 66, 31, 70, 2, 49, 39, 78, 82, 99, 13, 16, 57, 93, 7, 29, 20, 67, 9, 85, 64, 68, 75, 33, 18, 23, 72, 57, 54, 94, 36, 68,

69, 100, 17, 61, 39, 16, 11, 93, 13, 76, 30, 76, 95, 34, 1, 13, 68, 47, 28, 55, 21, 58, 38, 66, 97, 94, 78, 93, 20, 88]

34 numbers were not generated:

[4, 5, 6, 10, 12, 14, 15, 19, 22, 24, 26, 27, 32, 35, 37, 42, 44, 45, 46, 48, 51, 56, 59, 63, 71, 74, 77, 80, 81, 83, 84, 90, 92, 98]

Generated numbers:

[41, 81, 32, 27, 37, 51, 97, 37, 79, 41, 48, 58, 42, 12, 54, 35, 35, 88, 45, 73, 96, 46, 75, 89, 51, 76, 28, 35, 42, 68, 52, 64, 2, 45, 28, 81, 73, 16, 21, 44, 94, 47, 64, 64, 93, 19, 8, 94, 56, 78, 83, 9, 73, 100, 42, 78, 16, 94, 14, 69, 81, 61, 68, 37, 39, 30, 85, 60, 83, 62, 71, 86, 47, 49, 2, 79, 33, 75, 20, 70, 98, 23, 3, 33, 92, 36, 89, 21, 68, 39, 49, 29, 21, 20, 63, 11, 74, 68, 40, 55]

35 numbers were not generated:

[1, 4, 5, 6, 7, 10, 13, 15, 17, 18, 22, 24, 25, 26, 31, 34, 38, 43, 50, 53, 57, 59, 65, 66, 67, 72, 77, 80, 82, 84, 87, 90, 91, 95, 99]

Generated numbers:

[35, 89, 90, 29, 49, 40, 23, 35, 20, 97, 81, 88, 23, 27, 96, 68, 47, 9, 32, 45, 64, 66, 29, 88, 54, 85, 78, 59, 89, 97, 17, 85, 31, 25, 75, 86, 77, 10, 25, 49, 85, 30, 99, 81, 63, 39, 6, 34, 38, 52, 76, 34, 50, 78, 53, 86, 80, 20, 64, 61, 14, 38, 21, 41, 1, 88, 67, 96, 52, 16, 92, 60, 42, 59, 8, 90, 6, 72, 4, 57, 96, 9, 2, 57, 72, 4, 42, 14, 57, 59, 66, 55, 88, 29, 57, 41, 18, 39, 48, 3]

38 numbers were not generated:

[5, 7, 11, 12, 13, 15, 19, 22, 24, 26, 28, 33, 36, 37, 43, 44, 46, 51, 56, 58, 62, 65, 69, 70, 71, 73, 74, 79, 82, 83, 84, 87, 91, 93, 94, 95, 98, 100]

Generated numbers:

[56, 35, 52, 67, 50, 33, 30, 26, 82, 38, 94, 20, 2, 95, 62, 34, 59, 22, 19, 69, 26, 97, 52, 57, 57, 78, 30, 83, 77, 69, 94, 39, 69, 1, 84, 19, 63, 74, 59, 68, 95, 63, 71, 20, 57, 66, 75, 67, 93, 80, 59, 88, 1, 62, 21, 12, 40, 11, 43, 64, 11, 27, 50, 60, 62, 23, 13, 87, 97, 1, 62, 99, 54, 55, 83, 58, 71, 49, 69, 44, 67, 72, 99, 37, 54, 29, 65, 34, 7, 55, 59, 94, 22, 89, 6, 58, 22, 19, 69, 47]

38 numbers were not generated:

[3, 4, 5, 8, 9, 10, 14, 15, 16, 17, 18, 24, 25, 28, 31, 32, 36, 41, 42, 45, 46, 48, 51, 53, 61, 70, 73, 76, 79, 81, 85, 86, 90, 91, 92, 96, 98, 100]

Generated numbers:

[31, 52, 39, 21, 1, 27, 6, 55, 78, 56, 33, 17, 2, 65, 2, 90, 7, 89, 79, 74, 24, 25, 57, 93, 23, 96, 85, 4, 49, 50, 91, 21, 60, 6, 93, 10, 86, 14, 49, 57, 85, 33, 6, 42, 95, 69, 8, 48, 41, 76, 52, 33, 25, 92, 39, 64, 23, 62, 54, 73, 94, 86, 50, 81, 89, 33, 7, 16, 16, 90, 89, 100, 34, 9, 7, 9, 84, 19, 100, 72, 9, 20, 75, 95, 88, 33, 13, 61, 63, 46, 46, 30, 47, 99, 83, 13, 57, 2, 23, 11]

33 numbers were not generated:

[3, 5, 12, 15, 18, 22, 26, 28, 29, 32, 35, 36, 37, 38, 40, 43, 44, 45, 51, 53, 58, 59, 66, 67, 68, 70, 71, 77, 80, 82, 87, 97, 98]

Generated numbers:

[32, 88, 38, 97, 37, 1, 35, 83, 53, 46, 70, 58, 9, 55, 34, 40, 38, 98, 31, 87, 5, 1, 29, 79, 26, 27, 28, 28, 78, 53, 9, 62, 69, 27, 70, 99, 51, 51, 86, 40, 89, 80, 80, 36, 96, 92, 83, 80, 60, 73, 76, 24, 82, 42, 74, 69, 49, 41, 26, 94, 54, 41, 98, 82, 11, 90, 43, 59, 63, 15, 53, 60, 1, 96, 78, 79, 71, 17, 77, 78, 14, 19, 37, 76, 36, 26, 54, 55, 82, 56, 41, 45, 51, 11, 57, 97, 11, 64, 7, 78]

37 numbers were not generated:

[2, 3, 4, 6, 8, 10, 12, 13, 16, 18, 20, 21, 22, 23, 25, 30, 33, 39, 44, 47, 48, 50, 52, 61, 65, 66, 67, 68, 72, 75, 81, 84, 85, 91, 93, 95, 100]

Generated numbers:

[72, 11, 42, 80, 74, 20, 1, 27, 57, 85, 87, 62, 10, 25, 71, 17, 4, 68, 42, 20, 17, 32, 1, 67, 1, 97, 59, 97, 38, 27, 38, 9, 87, 100, 3, 65, 48, 94, 80, 87, 22, 23, 11, 3, 79, 46, 91, 34, 61, 9, 46, 76, 74, 67, 72, 48, 35, 6, 94, 97, 36, 23, 4, 9, 38, 33, 74, 32, 90, 51, 42, 12, 87, 84, 43, 16, 53, 76, 88, 22, 32, 60, 93, 89, 6, 87, 5, 61, 44, 3, 40, 70, 44, 49, 72, 53, 55, 20, 20, 78]

41 numbers were not generated:

[2, 7, 8, 13, 14, 15, 18, 19, 21, 24, 26, 28, 29, 30, 31, 37, 39, 41, 45, 47, 50, 52, 54, 56, 58, 63, 64, 66, 69, 73, 75, 77, 81, 82, 83, 86, 92, 95, 96, 98, 99]

The averaged missing generated was 36.7

Appendix 39

Gaussian Distribution probability density function

Equation of Gaussian Distribution

The probability density function (PDF) of a normal distribution is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where:

- x = Random variable
- μ = Mean (average)
- σ = Standard deviation
- $\pi \approx 3.14159$
- $e \approx 2.71828$ (Euler's number)

Standard Normal Distribution (Mean = 0, SD = 1)

When $\mu = 0$ and $\sigma = 1$, the equation simplifies to:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

This is called the **standard normal distribution**, commonly denoted as $N(0, 1)$.

Appendix 40

Gaussian Distribution and Human Heights

Mathematical Representation

The height of adult males in a given population (e.g., in the UK) follows a **normal distribution**:

$$H \sim N(\mu, \sigma^2)$$

Where:

- H represents height,
- μ is the **mean height** (e.g., 175 cm for adult males in the UK),
- σ is the **standard deviation** (e.g., 7 cm).

The probability density function (PDF) of height is:

$$f(h) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(h-\mu)^2}{2\sigma^2}}$$

For **UK adult males**:

$$f(h) = \frac{1}{7\sqrt{2\pi}} e^{-\frac{(h-175)^2}{2(7^2)}}$$

Standard Normal Distribution for Heights

If we **standardize** height using the **Z-score**:

$$Z = \frac{H - \mu}{\sigma}$$

Then, the distribution transforms into the **Standard Normal Distribution**:

$$Z \sim N(0, 1)$$

This means that:

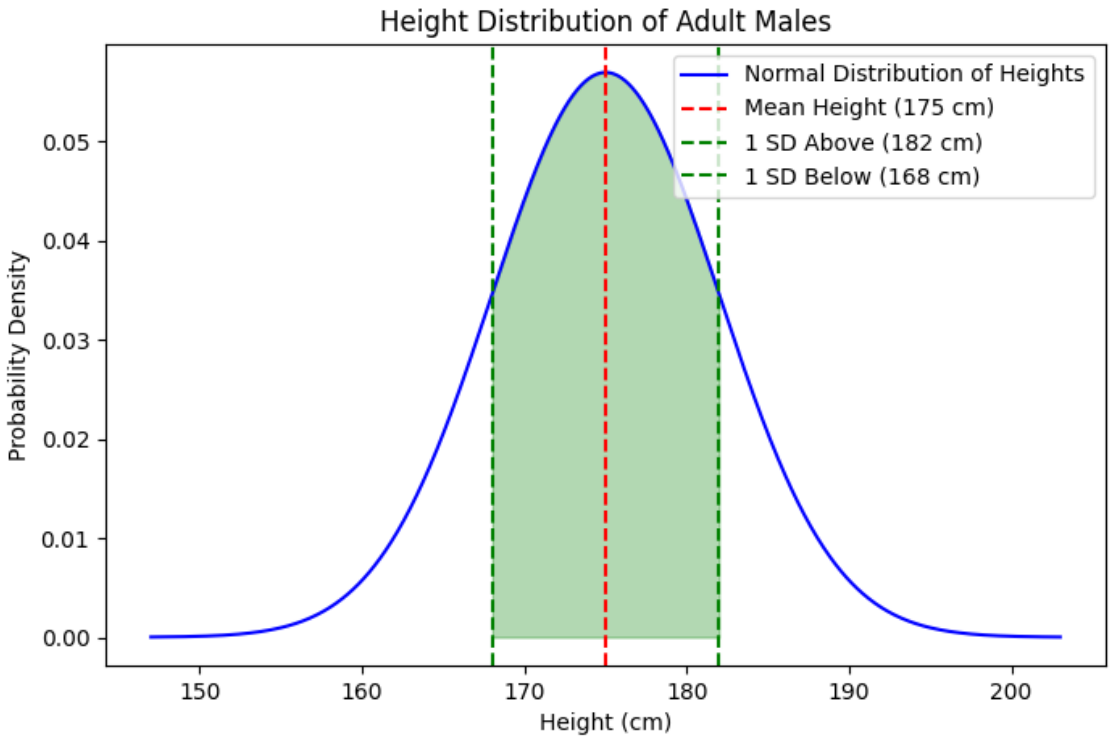
- If a person is **175 cm tall**, $Z = 0$ (they are exactly average).
- If a person is **182 cm tall**, $Z = \frac{182-175}{7} = 1$ (1 standard deviation above the mean).
- If a person is **168 cm tall**, $Z = \frac{168-175}{7} = -1$ (1 standard deviation below the mean).

From the **68-95-99.7 rule**:

- **68%** of people have heights between **168 cm and 182 cm** ($\mu \pm \sigma$).
- **95%** have heights between **161 cm and 189 cm** ($\mu \pm 2\sigma$).
- **99.7%** have heights between **154 cm and 196 cm** ($\mu \pm 3\sigma$).

Appendix 41

Height distribution (output and source code)

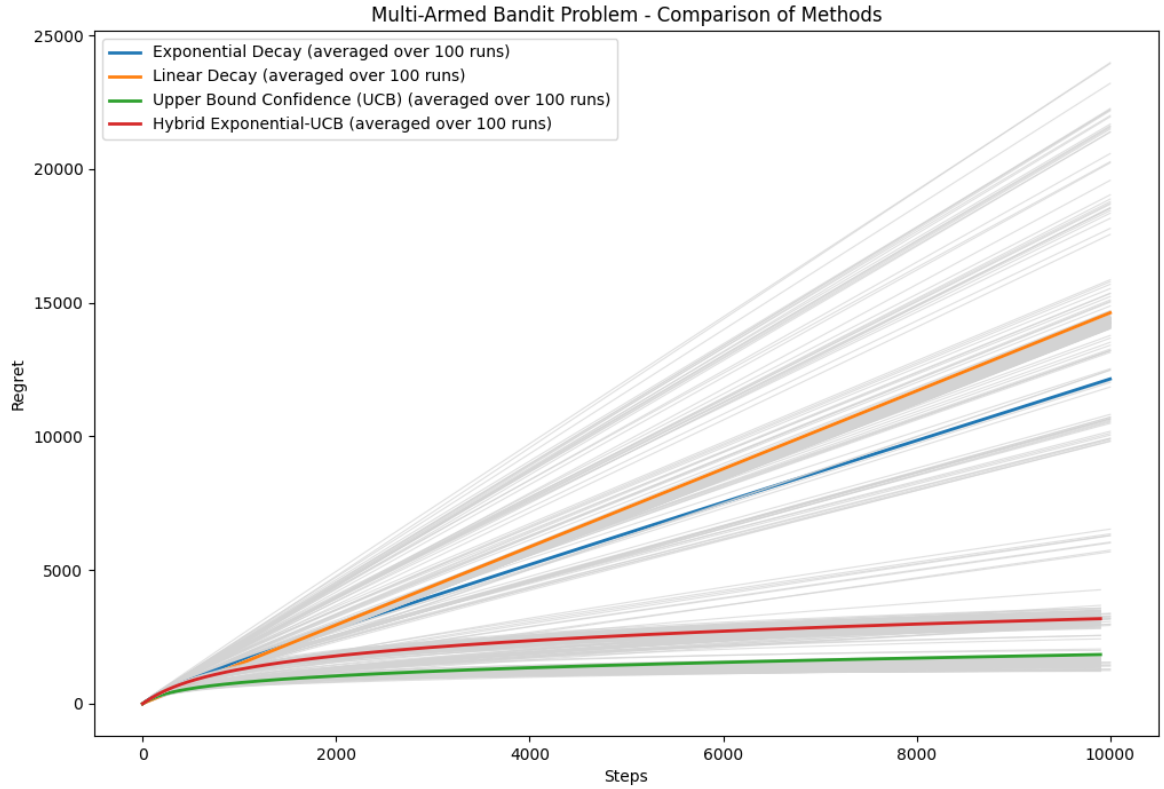



```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Mean and standard deviation for male height (in cm)
mu = 175
sigma = 7

# Generate height data
heights = np.linspace(mu - 4*sigma, mu + 4*sigma, 1000)
pdf = stats.norm.pdf(heights, mu, sigma)

# Plot the normal distribution
plt.figure(figsize=(8, 5))
plt.plot(heights, pdf, label="Normal Distribution of Heights", color="blue")
plt.axvline(mu, color='red', linestyle="dashed", label="Mean Height (175 cm)")
plt.axvline(mu + sigma, color='green', linestyle="dashed", label="1 SD Above
(182 cm)")
```

```
plt.axvline(mu - sigma, color='green', linestyle="dashed", label="1 SD Below
(168 cm)")
plt.fill_between(heights, pdf, where=((heights >= 168) & (heights <= 182)),
color='green', alpha=0.3)

plt.title("Height Distribution of Adult Males")
plt.xlabel("Height (cm)")
plt.ylabel("Probability Density")
plt.legend()
plt.show()
```

Appendix 42

Regret test output (spins = 10000, variance = 1, machines = 100, trials = 100)



Appendix 43

Performance Comparison (spins = 10000, variance = 1, machines = 100, trials = 100)

Exponential Decay -> Average Action Numerical Regret: 0.6606, Percentage Regret: 30.88%

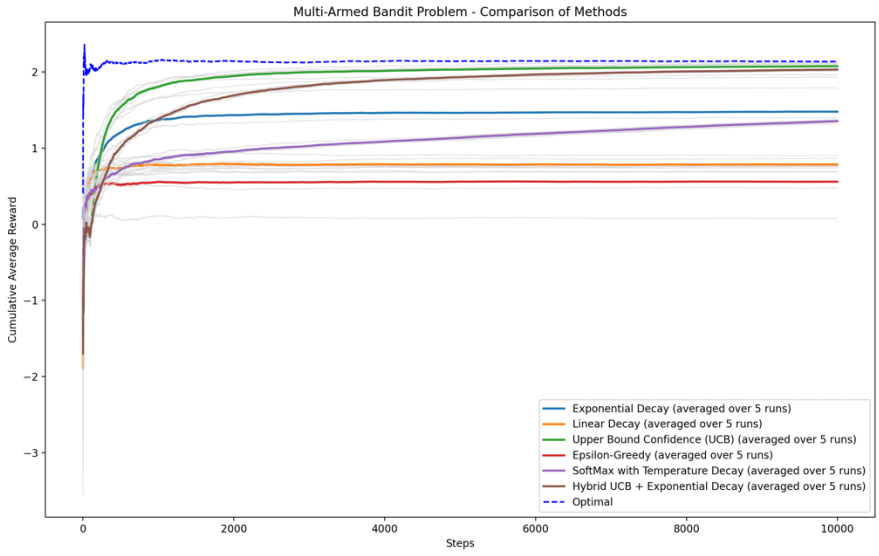
Linear Decay -> Average Action Numerical Regret: 1.3527, Percentage Regret: 63.23%

Upper Bound Confidence (UCB) -> Average Action Numerical Regret: 0.0634, Percentage Regret: 2.96%

Epsilon-Greedy -> Average Action Numerical Regret: 1.5807, Percentage Regret: 73.88%

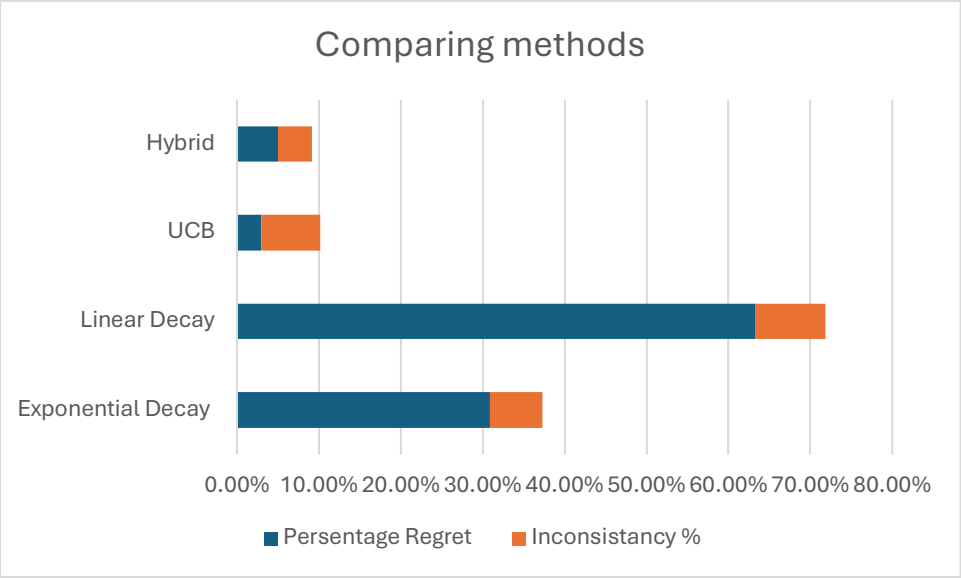
SoftMax with Temperature Decay -> Average Action Numerical Regret: 0.7842, Percentage Regret: 36.65%

Hybrid UCB + Exponential Decay -> Average Action Numerical Regret: 0.1077, Percentage Regret: 5.03%



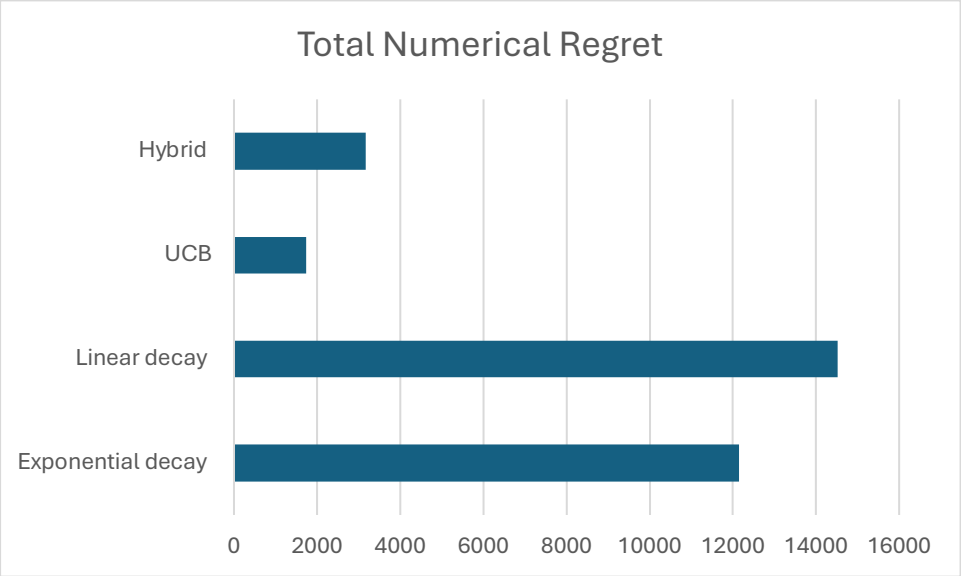
Appendix 44

Graph comparing results



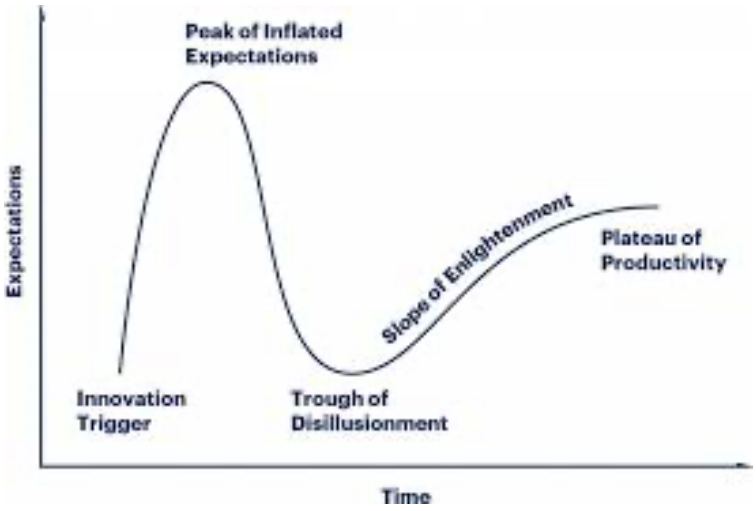
Appendix 45

Graph showing total regret comparison



Appendix 46

Gartner Hype Cycle



[Introduction to the Gartner Hype Cycle – BMC Software | Blogs]