

SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++

BY

LIM YIK HENG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

REPORT STATUS DECLARATION FORM

Title: ____Solving Geometric Quadratic Stochastic Operators Using C++

Academic Session: __Jan 2024_____

I _____ LIM YIK HENG _____
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)

Siti Nurlaili

(Supervisor's signature)

Address:

_1, Persiaran Bandar Baru Tambun 15,____
_Desa Tambun Indah,_____
_31400, Ipoh, Perak_____

_ Dr. Siti Nurlaili Karim ____
Supervisor's name

Date: _23/4/2024_____

Date: __23/4/2024_____

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

**FACULTY/INSTITUTE* OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN**

Date: ____1/4/2024_____

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that **LIM YIK HENG** (ID No: **20ACB03395**) has completed this final year project/ dissertation/ thesis* entitled “ **Solving Geometric Quadratic Stochastic Operators Using C++** ” under the supervision of Dr. Siti Nurlaili Karim (Supervisor) from the Department of Computer Science , Faculty/Institute* of Information and Communication Technology , and - (Co-Supervisor)* from the Department of - , Faculty/Institute* of - .

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



Scanned with CamScanner

(LIM YIK HENG)

DECLARATION OF ORIGINALITY

DECLARATION OF ORIGINALITY

I declare that this report entitled “**SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____  _____

Name : _____ LIM YIK HENG _____

Date : _____ 1/4/2024 _____

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr. Siti Nurlaili Karim who has given me this bright opportunity to engage in a project which may benefit plenty of mathematicians in the future. It is my first step to establish a career in IT development field. A million thanks to you.

To a very special person in my life, Zoe Yau, for her patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

In this research paper, an investigation is carried out to analyze the performance characteristics of C++ and Maple programming in the context of solving specific classes of Geometric Quadratic Stochastic Operators (GQSOs). The primary objective is to discern and evaluate the computational efficiency and reliability of these programming for addressing GQSOs, thereby offering valuable insights into their respective strengths and limitations. To meet this objective, a specific C++ program is designed to handle some classes of GQSOs that had chosen which are the ones solved earlier using the MAPLE programming. We then compare the results from our C++ program with the results from the Maple environment. C++ is chosen because it is less advance compared to Maple, which is a language that may confuse researcher or mathematician who lack of knowledge in programming, and Maple may take a long period of time to solve GQSO and obtain the result. In terms of the methodology, Visual Studio 2019 is used to create a tailored C++ program designed to effectively tackle the challenges posed by GQSOs. The results we obtained were stored into structured text files. Then, Gnuplot use the data as input to plot graphs for further analysis. This paper's contribution is introducing a specialized C++ program that offers convenience and efficiency to mathematicians and researchers in the field. This program serves as an accessible alternative to using MAPLE programming, making it viable for a wider audience, even those less experienced in programming. In conclusion, this research provides a well-structured analysis comparing C++ and MAPLE programming in solving various classes of GQSOs as well as a functional C++ program.

Keywords: C++, Maple, quadratic stochastic operators, Geometric distribution.

TABLE OF CONTENTS

TITLE PAGE	i
REPORT STATUS DECLARATION FORM	ii
FYP THESIS SUBMISSION FORM	iii
DECLARATION OF ORIGINALITY	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii-ix
LIST OF FIGURES	x-xi
LIST OF TABLES	xii
LIST OF ABBREVIATIONS	xii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	2-3
1.2 Objectives	3-4
1.3 Project Scope	4
1.4 Contributions	4-5
1.5 Report Organization	5
CHAPTER 2 LITERATURE REVIEW	6
2.1 Previous study on C++ Programming Language and Maple Programming	6
2.1.1 Loop Styles in Modern C++	6-7
2.1.2 Solving QSOs with Maple Programming	7
2.1.3 Advantages of C++ Programming	8
2.1.4 Visualization of Data in C++	8-9
2.2 Limitation of Previous Studies	9-10
2.3 Proposed Solutions	10-11
2.4 Summary	12

CHAPTER 3 SYSTEM METHODOLOGY/APPROACH	13
3.1 System Design Diagram/Equation	13-14
3.1.1 System Architecture Diagram	14-15
3.1.2 Use Case Diagram and Description	15-16
3.1.3 Activity Diagram	16-17
3.2 Timeline	17-20
CHAPTER 4 SYSTEM DESIGN	21
4.1 System Block Diagram	21
4.2 System Components Specifications	22
4.2.1 Function of PlotOrbit and Visualization of Plotorbit	22-25
4.2.2 Function of Cobweb and Visualization of Cobweb	25-28
CHAPTER 5 SYSTEM IMPLEMENTATION	29
5.1 Software and Hardware Setup	29
5.2 System Operation	30-32
5.3 Implementation Issues and Challenges	33
5.4 Concluding Remark	33
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	34
6.1 Testing Setup and Result	34-40
6.2 Project Challenges	40
6.3 Objectives Evaluation	40
6.4 Concluding Remark	41
CHAPTER 7 CONCLUSION AND RECOMMENDATION	42
7.1 Conclusion	42
7.2 Recommendation	42
REFERENCES	43
WEEKLY LOG	44-53
POSTER	54

PLAGIARISM CHECK RESULT

55-59

FYP2 CHECKLIST

60

LIST OF FIGURES

Figure Number	Title	Page
Figure 3.1	Procedural programming	14
Figure 3.1.1	System architecture diagram	14
Figure 3.1.2	Use case diagram of the system	15
Figure 3.1.3	Activity diagram of the system	16
Figure 3.2.1	Timeline from Week 1 to Week 4	17
Figure 3.2.2	Timeline from Week 5 to Week 8	18
Figure 3.2.3	Timeline from Week 9 to Week 11	18
Figure 4.1	General system process design diagram	21
Figure 4.2.1	C++ code for solving A00 equation	22
Figure 4.2.2	Calculation of coefficients	22
Figure 4.2.3	The $f(x)$ function	22
Figure 4.2.1.1	Call of PlotOrbit function	22
Figure 4.2.1.2	PlotOrbit function	23
Figure 4.2.1.3	Main calculation part of PlotOrbit	23
Figure 4.2.1.4	Storing and visualization of result of PlotOrbit	23
Figure 4.2.1.5	Visualization of result of PlotOrbit	24
Figure 4.2.1.6	Visualization of result of PlotOrbit for first 100 iterations	24
Figure 4.2.1.7	Motion graph of PlotOrbit for first 100 iterations	24
Figure 4.2.2.1	Calculation part of Cobweb (horizontal)	25
Figure 4.2.2.2	Storing result of Cobweb (horizontal)	25
Figure 4.2.2.3	Calculation part of Cobweb (vertical)	26
Figure 4.2.2.4	Storing result of Cobweb (vertical)	26
Figure 4.2.2.5	Visualization of Cobweb	27
Figure 4.2.2.6	Motion graph of Cobweb	27
Figure 4.2.2.7	Function of Cobweb	27
Figure 5.2.1	Creation of folder	30
Figure 5.2.2	Error handling of creation of folder	30
Figure 5.2.3	Update value or Default	30

Figure 5.2.4	Update value	31
Figure 5.2.5	Display update values	31
Figure 5.2.6	Display default values	31
Figure 5.2.7	PlotOrbit and Cobweb	32
Figure 5.2.8	Restart of system	32
Figure 6.1.1	Parameter in Maple	34
Figure 6.1.2	Seed value of PlotOrbit and Cobweb	34
Figure 6.1.3	Parameter in C++ system	35
Figure 6.1.4	Comparison of Result	36
Figure 6.1.5	Cobweb from Maple	37
Figure 6.1.6	Cobweb from Gnuplot	37
Figure 6.1.7	PlotOrbit from Maple	38
Figure 6.1.8	PlotOrbit from Gnuplot	39
Figure 6.1.9	Comparison of execution time	39

LIST OF TABLES

Table Number	Title	Page
Table 5.1	Specifications of laptop	28

LIST OF ABBREVIATIONS

<i>GQSO</i>	Geometric Quadratic Stochastic Operator
-------------	---

CHAPTER 1 INTRODUCTION

The exploration of Quadratic Stochastic Operators (QSOs) finds its origins in the research conducted by Bernstein in 1920s [1]. Quadratic Stochastic Operators (QSOs) also known as “evolutionary Operator” are pivotal mathematical tools have been acknowledged as a major basis in the exploration of dynamic characteristics and modelling across numerous fields [2]. In recent years, the application of these operators has witnessed a notable increase due to their functional capabilities. These advancements have fuelled a heightened interest in approaches to solve these operators as well as improving their efficiency.

Geometric Quadratic Stochastic Operators (GQSOs) play a crucial role in understanding complex systems affected by randomness. However, solving GQSOs becomes challenging with large datasets and complex probabilities. Traditional methods, which use Maple programming when solving some classes of GQSOs are insightful but struggle with User-friendliness and computational speed. This situation calls for new and effective ways to solve GQSOs. This study is driven by the goal of narrowing the gap between the theoretical potential of GQSOs and their practical applicability. Leveraging modern computing techniques, harnessing the computational power of C++ programming emerges as a promising strategy to enhance GQSO solutions. C++ is known for its speed and adaptability, making it a suitable candidate for effectively addressing these complex mathematical models. By combining C++ with the intricacies of GQSOs, our aim is to speed up solutions while ensuring accuracy for real-world use.

To gain new insights into the complex dynamics of stochastic systems, we explore the interface between GQSOs and C++ programming in this work. This voyage highlights the compatibility between mathematical theory and actual code, as well as the possibilities for streamlined solutions. This research aims to improve our understanding of GQSOs and, in turn, make a significant contribution to the field of computational mathematics and its practical applications.

1.1 Problem Statement and Motivation

As stated earlier, there has been a noticeable increase in the use of these operators in recent times. Some specific classes of GQSOs have been chosen for solving, and this process involves a significant number of calculations. These calculations are extensive enough to require computer assistance. Traditionally, Maple, a sophisticated tool for mathematical problem-solving, has been used to solve GQSOs. While Maple is unquestionably useful, its computing performance is lacking, particularly for moderately sized datasets and considerably more so for larger datasets. In addition, those without a solid background in programming or who are unfamiliar with key terms may find it difficult to understand the intricacies of Maple programming. This could pose a challenge for mathematicians and researchers working in this area, particularly when trying to fully understand and solve these operators. This issue emphasises the need for an alternate strategy that strikes a balance between computing speed, accuracy, and accessibility. The study of C++ programming takes centre stage here. Researchers' capacity to efficiently solve GQSOs may be improved by the speed and versatility of C++. This switch from Maple to C++ streamlines processes, making GQSO analysis more user-friendly and effective, promoting more fluid research development and insightful understanding in this area.

Furthermore, solving certain types of QSOs has a significance that goes beyond simple mathematical exercises. It holds the key to understanding how many different natural and physical events behave. Researchers can obtain important insights into the underlying dynamics of these events and, in turn, forecast their future behaviours by effectively solving certain kinds of QSOs. This capacity for prediction has broad consequences, from comprehending sophisticated biological systems to predicting the behaviour of complex physical processes. As a result, utilising cutting-edge computational techniques like C++ to try to solve these QSOs is not only a theoretical endeavour but also a useful tool to solve real-world problems that will help us understand more about the natural world and expand our understanding in many different scientific fields.

The main objective of this project is to present a functional C++ programme that has been created expressly to meet the complex problems posed by GQSOs. This effort is focused on two keys that form the basis of the investigation. The first aspect focuses on the difficulty of using the suggested C++ programme to address GQSOs. Due to the underlying mathematical complexity and probabilistic linkages, GQSOs necessitate a methodical approach to be solved. This project tries to decipher the complexity related to GQSOs and give a streamlined route for

their resolution by developing a purpose-built C++ programme. The goal of the research is to advance our knowledge of the fundamental dynamics of stochastic systems and to pave the way for practical applications in a wide range of sectors. Utilising the computational power of C++ programming is the second focus of the project. For handling complex calculations, C++ is known for its effectiveness, speed, and versatility. The goal of the thesis is to use these characteristics to enhance C++'s processing power when dealing with GQSOs. This entails bridging the gap between theoretical mathematics and real-world application, providing a way to arrive at effective and precise solutions for these complex operators.

In essence, the project sets out on two explorations: exploring the complexities of GQSOs and utilising the computing power of C++ programming. By combining these two aspects, the research aims to develop both computational methods and mathematical theory, ultimately serving as a useful tool for academics looking for practical answers to complicated stochastic systems.

1.2 Project Objectives

The primary goal of this project is to develop a C++ programming code capable of effectively solving GQSOs while also saving the results in a text file. This objective guides the development efforts, aiming to create a convenience and user-friendly computational tool which allow the users to change the variable of the GQSOs by modifying the input of parameter. It also facilitates post-processing such as accuracy and consistency and insights through the stored data. Storing the results in the text file serves a dual purpose: firstly, it provides researchers with a detailed record of the generated outcomes, serving as a valuable reference for analysis and verification. Secondly, this stored data acts as input parameters for Gnuplot, a data visualization tool, enhancing the project's overall utility. Through the utilization of Gnuplot, users can generate graphical representations of the results, aiding in a deeper understanding and insightful visualization of the complex QSO data, ultimately contributing to more effective decision-making and problem-solving in stochastic modelling and analysis.

Within the main project goal, the effort is divided into two deciding sub-objectives. The first sub-objective aims to generate results through a huge number of calculations, highlighting the importance of the efficiency of the code in handling complex and repetitive computations. Ensuring this level of precision is crucial not only for accurate results in solving GQSOs but also for guaranteeing the reliability of the code performance in resource-constrained

CHAPTER 1

environments. The second sub-objective focuses on creating graphical plots to visually represent the obtained results. By offering these clear visual representations, the project aims to make the data more understandable and user-friendly, simplifying the interpretation and analysis process. This approach ultimately enhances the project's overall effectiveness in dealing with complex GQSOs, enabling researchers and analysts to gain deeper insights from the generated outcomes.

1.3 Project Scope

The proposed project title is both clear and specific, effectively communicating its primary goal which is constructing a C++ code that can solve some classes GQSOs effectively and efficiently. It provides a concise understanding of the aim of the project and expected outcomes. The project scope consists of the development of a C++ programming code designed to empower users to tackle specific classes of GQSOs defined on a countable state space. These issues typically involve a lot of calculations done repeatedly and call for the creation of graphs to illustrate the results clearly. The main approach involves developing a streamlined program capable of effectively solving these intricate mathematical issues. Following the outcome of this project, the program's efficiency will be evaluated by analysing the simulation results, which will act as evident measures of its accomplishment. Ultimately, a C++ code will be constructed which ensures that users will possess a valuable instrument to tackle intricate GQSOs, simultaneously offering transparent visual representations of the outcomes, thereby simplifying analysis processes.

1.4 Contributions

This project's main contribution is the development of an effective and user-friendly C++ programming code designed to tackle the difficulties involved in solving GQSOs. This program makes the topic more reliable for academics and mathematicians while also illustrating the practicality of GQSO resolution. The importance of the programming code rests in its capacity to offer a simpler and more efficient method of resolving GQSOs. The program speeds up the resolution process and produces results that are more accurate and efficient by utilizing the computational capabilities of C++. By speeding up the analysis stage and providing a useful tool for acquiring results, this development immediately aids researchers and mathematicians in the field. The simplicity of C++ also makes it accessible to a wider range of people, including those who lack in-depth programming knowledge. This openness enables researchers and

mathematicians to produce data and carry out visualization for in-depth study without being constrained by technical limitations. In essence, this work provides a practical solution that not only shows how successful addressing GQSOs is, but also supports a more open and effective research environment. By combining computational efficiency and user-friendliness, the study enables a deeper analysis of GQSOs and their implications across several fields.

1.5 Report Organization

The details of this research are shown in the following chapters. In Chapter 2, an in-depth exploration of relevant backgrounds is presented, covering a detail review of Maple programming and C++. The analysis aims to clarify the functionalities, strengths, and weaknesses existed in both programming paradigms. Then, a preliminary idea is provided on the methodology and technologies employed to address the GQSO challenge as well as the system design and architecture of the system in Chapter 3. Chapter 4 provides an overview of the preliminary work and results of the project, delving into the feasibility of future development. This section highlights a detailed discussion on the methods employed to address the GQSOs using C++. Moving forward, Chapter 5 will draw conclusions, offering insights into the overall performance of the project and summarizing key findings.

CHAPTER 2 Literature Review

2.1 Previous study on C++ programming language and Maple programming.

2.1.1 Loop Styles in Modern C++

Understanding the looping styles in C++ is crucial for maximizing the performance of complex computations like solving GQSOs problems, which often involve extensive looping. These looping styles, including index-based loops, iterators, and range-based loops, play a key role in writing efficient code. By choosing the right loop style and optimizing code for clarity and maintainability, developers can harness the computational power required for GQSO tasks effectively, ensuring that the code runs efficiently without unnecessary bottlenecks. This understanding becomes particularly valuable when dealing with computationally intensive tasks, where even small performance improvements in loop constructs can make a significant difference in overall execution speed and resource utilization.

Paterno [4] proposed a paper in 2014 that studied the robustness and computational efficiency of C++. The article investigates the computational efficiency of various loop styles in Modern C++. It explores four primary loop constructs: a non-idiomatic index-based loop using an int counter, an idiomatic index-based loop with an unsigned long counter, an iterator-based loop, and a range-for loop. These loop styles are evaluated in terms of code size and computational efficiency. The non-idiomatic index-based loop is less efficient due to its use of an int counter. Based on study in [4], its code size is 69 bytes, and the timing results showed a mean execution time of 22.98 ± 0.80 million ticks. In contrast, the idiomatic index-based loop with the correct type for the loop counter, unsigned long, demonstrated slightly better efficiency with a code size of 63 bytes and a similar timing of 22.80 ± 0.57 million ticks.

The iterator-based loop and range-for loop both provided clear and flexible looping interfaces. Their code sizes were smaller than the non-idiomatic index-based loop, and their execution times were comparable, with iterator-based loop taking 22.82 ± 0.79 million ticks [4]. These findings dispel concerns about the performance of iterator-based loops, suggesting that they are an efficient choice. Additionally, Paterno [4] highlights the use of the Standard Library algorithm, as a highly compact and efficient option. This algorithm, purpose-built for summation tasks, produced the smallest code size and exhibited efficiency similar to the hand-written loops.

In conclusion, the article underscores that while there are minor differences in code size and performance between loop styles, the choice of loop construct should prioritize code clarity and maintainability. Developers are encouraged to embrace idiomatic C++ constructs and

Standard Library algorithms when available, as they offer clear, efficient, and maintainable code without significant computational efficiency trade-offs.

2.1.2 Solving QSOs with Maple Programming

Solving quadratic stochastic operators (QSOs) has attracted considerable research attention, mainly in theoretical domains. Researchers have established the theoretical groundwork, delving into the fundamental principles and characteristics that govern the behaviour of QSOs. However, when dealing with practical and real-world QSO scenarios, the need for numerical solutions becomes evident. This is where computational analysis comes into play, serving as a crucial bridge between theory and practical application.

To address the challenges posed by complex QSO problems, Mukhamedov and Embong [5] employ specialized software tools like Maple. Maple is a symbolic computation language, enabling it to solve problems symbolically rather than solely relying on numerical solutions, setting it apart from many other programming languages [6]. Utilizing software platforms like Maple empowers researchers to construct and develop programming code, facilitating the extraction of numerical solutions and the creation of graphical representations. Maple offers a vast library of mathematical functions and tools, equipping researchers with a comprehensive toolkit for tackling intricate computations and data visualization tasks. This extensive mathematical support streamlines the process of working with complex mathematical equations, enabling researchers to perform in-depth calculations efficiently. Additionally, Maple's capabilities extend beyond numerical solutions, allowing for symbolic computations, algebraic simplifications, and mathematical modelling, making it a valuable resource for researchers seeking to address a wide range of mathematical challenges. As a result, it offers the computational capabilities required to tackle intricate QSOs issues effectively.

Nonetheless, this process comes with its own set of challenges. Researchers often find themselves in the position of constructing complex programming code from scratch. Creating the necessary code for solving QSOs can be challenging and requires researchers to learn how to use specific functions and libraries effectively. This learning process often comes with a steep learning curve. As a result, generating numerical results and visual representations like graphs can be quite time-consuming.

2.1.3 Advantages of C++ Programming

As mentioned, Maple programming can solve the QSOs well, but the computational time may be a concern when comes to real-world applications. Rassokhin [7] proposed a paper that study the structure of C++ programming and gives a clear image about its advantages compare to other programming languages. The advantage of using a compiled language like C or C++ is that it produces machine code optimized for a specific target platform, which can result in faster and more efficient execution of programs [7]. Compiled code is platform-specific and does not require an interpreter or runtime environment, making it suitable for system programming and high-performance computing tasks. This efficiency and direct machine code execution are advantageous when performance is critical, such as in embedded systems, real-time applications, or resource-intensive tasks like scientific computing and game development.

Rassokhin [7] also proposed that C++ is a user-friendly and efficient programming language. The context of software development, it is a common practice to apply C or C++ to applications, even there are other better choices of programming languages. This mixture plays a main role in achieving high performance and fine-tuning hardware control in critical parts of the software. For instance, Python has an absolute advantage when dealing with prototyping and scripting, but C++ and C are needed when dealing with heavy computational task such as large amount of looping during the solving QSOs process. Besides, Modern C++ versions make it easier to create portable code that makes use of modern multicore computer architectures [7]. These versions provided the tools for creating and managing these threads, as well as for allocating memory access among multiple concurrently operating threads. To fully leverage a computer's computational capabilities, it is crucial to employ a programming language that can effectively optimize for the hardware.

2.1.4 Visualization of Data in C++

As stated in the project objectives, data visualization is a crucial feature that must be developed in the final system. To fulfill this requirement, several visualization tools are considered to identify the most suitable approach for efficiently and effectively visualizing the data. In [8], it was noted that matplotlib is one of Python's libraries. Although matplotlib is a powerful tool for data visualization, but integrating matplotlib with a C++ codebase necessitates inter-language communication, typically achieved through wrappers like Pybind11 or by embedding Python within C++. Certainly, integrating motion graph generation into the system with matplotlib may introduce additional complexity to the development phase.

Other than that, interfacing with Python from C++ can introduce performance overhead due to the overhead of Python's interpreter and the communication between the two languages. As the project aims to minimize computer resource usage, it is important to avoid unnecessary complexities. Moreover, matplotlib excels in generating 2D plots [8]. Hence, its suitability for this project, which involves changes over iterations, may be limited. Given the significance of motion graphs as a key outcome, prioritizing their development is paramount.

2.2 Limitation of Previous Studies

In section 2.1.1, which related to previous research on C++ programming focusing on looping styles and computational time, there are a couple of limitations worth noting. First, the study employed a relatively loop count of 1000, which, in the world of IT, is not considered a particularly large number. Furthermore, the study in [3] clarifies that the purpose of the iteration is to illustrate how the probability distribution of population evolves over time, with a series of iterative steps starting from an initial state and continuing through subsequent generations. Consequently, this sample size may not be adequate for capturing the full spectrum of potential variations and outliers in execution time. This limitation becomes particularly concerning when considering that the number of loops required for solving QSOs typically exceeds 1000. As a result, the identified limitation could potentially impact the program's efficiency in handling larger workloads in the future. Another consideration is the hardware used in the study, which was an Intel I7 @2.7 GHz processor, representing a relatively newer version. This choice of hardware may introduce concerns when extending the results to different environments, particularly on laptops equipped with older processor versions. There are concerns about the generalizability of the findings across a wider variety of hardware configurations due to the potential effects of the variations in processor specs and capabilities on the program's performance and execution times.

In section 2.1.2, which is related to solving QSOs with Maple programming, it is worth mentioning that Maple can present challenges for certain users and scenarios. While Maple offers substantial benefits for tackling complex mathematical problems, it also presents certain limitations. Firstly, the software's computational capabilities, especially when dealing with symbolic computations and intricate mathematical modeling, can be resource intensive. This may result in longer processing times, particularly on older or less powerful computer systems, impacting the efficiency of researchers working under such constraints. Secondly, the extensive functionality and complexity of Maple can be overwhelming for new users, requiring a steep

CHAPTER 2

learning curve to use it to its full extent. This process may deter researchers who seek more straightforward and user-friendly tools for their numerical and mathematical tasks. Additionally, it's important to note that Maple is a commercial software product, and its cost can be a limitation for individuals or institutions with budget constraints.

Section 2.1.3 highlights C++ as a convenient programming language, but when applied to the QSOs, certain challenges emerge. QSOs involve plenty of mathematical symbols and operations that are not the primary focus of C++ programming, making it a lesser choice for dealing with these problems. This project aiming to create functional QSOs solving programs may face a difficult learning phase, as a substantial amount of time is required to understand the mathematical structures and principles behind QSOs. This initial learning phase can be time-consuming, potentially delaying the development process. While C++ offers great flexibility and computational power, its strength lies more in low-level hardware access and performance optimization rather than in seamlessly handling mathematical symbolism.

In 2.1.4, it was noted that while matplotlib are powerful, it also come with several drawbacks. Since the system will be developed using C++, it is crucial to recognize that libraries like matplotlib are not inherently included. This implies that additional effort will be required to integrate matplotlib into the system. Such integration efforts may extend the development phase, potentially causing delays in other phases if compatibility issues arise. Moreover, dynamic presentation of data stands out as a crucial outcome for the project. However, since matplotlib excels primarily in 2D plotting, it may not fully meet the project's requirements for dynamic data visualization. This highlights the need to explore alternative solutions that can effectively support the project's objectives for dynamic data presentation.

2.3 Proposed Solutions

To address the limitation mentioned in section 2.1.1, the looping number will be set to 1000 as a starting point for evaluating the functionality and performance of the code, ensuring there is no overburdening of the system. The results will be closely scrutinized for accuracy and potential issues. Subsequently, the looping number will be increased to 10000 to simulate a more demanding computational workload, enabling us to observe changes in computational time and assess result quality under heavier processing. By comparing outcomes between the two settings, the aim is to gain insights into its scalability and efficiency. If there are any negative issues, optimization of the codes will be done to maintain reasonable execution times

CHAPTER 2

while preserving result accuracy. Overall, this systematic approach ensures that the code can effectively manage different workloads and provide reliable and efficient results.

To overcome the limitation mentioned in section 2.1.2, a C++ programming code will be developed to enable users to solve a specific class of GQSOs defined on a countable state space. This code will accommodate the generation of results from a substantial number of iterations and facilitate the visualization of these results through graphical plots. The project aims to construct efficient algorithms, a simple and easy-to-understand user interface for parameter input which allow users to put different value for different situation. Then, the result will be store in text file as a reference and easy to pass the result into Gnuplot for continuing visualization of the data. This user-centric approach aims to provide a versatile tool for users to address complex problems while offering support and feedback mechanisms to enhance usability and effectiveness.

To address the limitation identified in section 2.1.3, a strategic approach involves utilizing a sample Maple programming code that has effectively solved certain classes of GQSOs as a reference point when constructing the C++ code. This reference code serves as a valuable resource by boosting the development process through the understanding of the mathematical structure, algorithms, and problem-solving techniques embedded in the Maple code. It enables developers to efficiently adapt and modify the existing solutions, significantly reducing implementation time while ensuring accuracy through result comparison. This approach accelerates program delivery and facilitates a seamless transition from Maple to C++, though careful consideration of language-specific nuances and optimization remains imperative.

To address the data visualization challenges, selecting a new visualization tool is imperative to meet the project's objectives effectively. Gnuplot emerges as a promising tool for the project, offering motion graph generation capabilities that align with our needs. Unlike matplotlib, which relies on Python and may be perceived as more advanced, Gnuplot's syntax is often considered more straightforward and easier to understand. This simplicity could streamline the development process and facilitate quicker implementation of dynamic data visualization features. Also, it takes care of those non-IT researcher as they may look into the command and gaining a better insight of the outcomes. Therefore, leveraging Gnuplot could prove advantageous in achieving the project's objectives efficiently and effectively.

2.4 Summary

Drawing from the review, a comprehensive study of the strengths and weaknesses that are existed in both C++ and Maple Programming has been conducted. The analysis reveals that while Maple provides a more conducive environment for GQSO resolution, its user-friendliness may pose challenges for individuals lacking a programming background. Furthermore, Maple requires extended computational times in GQSO problem-solving, even with a modest loop iteration count set at 1000, thus raising concerns regarding practical real-life applicability.

In contrast, the evaluation of C++ underscores its outstanding processing power, particularly well-suited for handling heavy workloads such as extensive looping—an essential aspect in GQSO problem resolution. This observation positions C++ as a potent environment for addressing GQSOs efficiently, particularly in scenarios necessitating expedited computational performance. Then, a more suitable visualization tool has been identified, one that aligns well with the project's objectives and can effectively support our goals.

Chapter 3 System Methodology/Approach

3.1 System Design Diagram/Equation

The developed system consists of three distinct phases: algorithm construction, result storage, and data visualization. The designated hardware for this system is a laptop, as previously specified in the relevant section. The related software components will be installed on the laptop to facilitate the entire development process seamlessly. This report will primarily emphasize the detailed system design specifically to the developed code, offering a focused exploration of the coding aspects central to the system's functionality.

Here is the format of GQSOs that will be solved by the system:

$$A00 = \sum_{k=p0}^{p1} (1 - r0) \cdot r0^k \quad (1)$$

$$A01 = \sum_{k=p0}^{p1} (1 - r1) \cdot r1^k \quad (2)$$

$$A02 = \sum_{k=p0}^{p1} (1 - r2) \cdot r2^k \quad (3)$$

$$A10 = \sum_{k=p2}^{p3} (1 - r0) \cdot r0^k \quad (4)$$

$$A11 = \sum_{k=p2}^{p3} (1 - r1) \cdot r1^k \quad (5)$$

$$A12 = \sum_{k=p2}^{p3} (1 - r2) \cdot r2^k \quad (6)$$

where

$r0, r1, r2$ are double, where > 0 and < 1

$k, p0, p1, p2, p3$ are integer

Figure below shows the concept of procedural programming.

```

void updateUserValues(int& loop_val, int& digit_val, int& p0_val, int& p1_val, int& p2_val, int& p3_val, double& r0_val, double& r1_val, double& r2_val);
double f_function(double x, double a1, double b1, double c1);
double derivative_function(double x, double a1, double b1, double c1);
void plotOrbit(function<double(double)> expression,int iterations);
void horizontal(double seed, double a1, double b1, double c1, int iterations);
void vertical(double seed, double a1, double b1, double c1, int iterations);
void plotCOBWEB(double a1, double b1, double c1, int loop);
void Orbit();
void Orbit_100();
void Orbit_motion();
void Initialization();
void temp();
void cobweb();
void cobweb_motion();
void update_loop(int& loop_val, int& digit_val);
void update_r(double& r0_val, double& r1_val, double& r2_val);
void update_p(int& p0_val, int& p1_val, int& p2_val, int& p3_val);

```

Figure 3.1: Procedural programming.

The figure illustrates the methodology approach of the system, characterized by procedural programming. Each function is constructed independently, ensuring that the failure of one function does not adversely affect others. This modular approach facilitates easier debugging and modification, as functions are called sequentially rather than all at once. This streamlines the evaluation process for each function, enhancing the overall reliability of system and ease of maintenance.

3.1.1 System Architecture Diagram

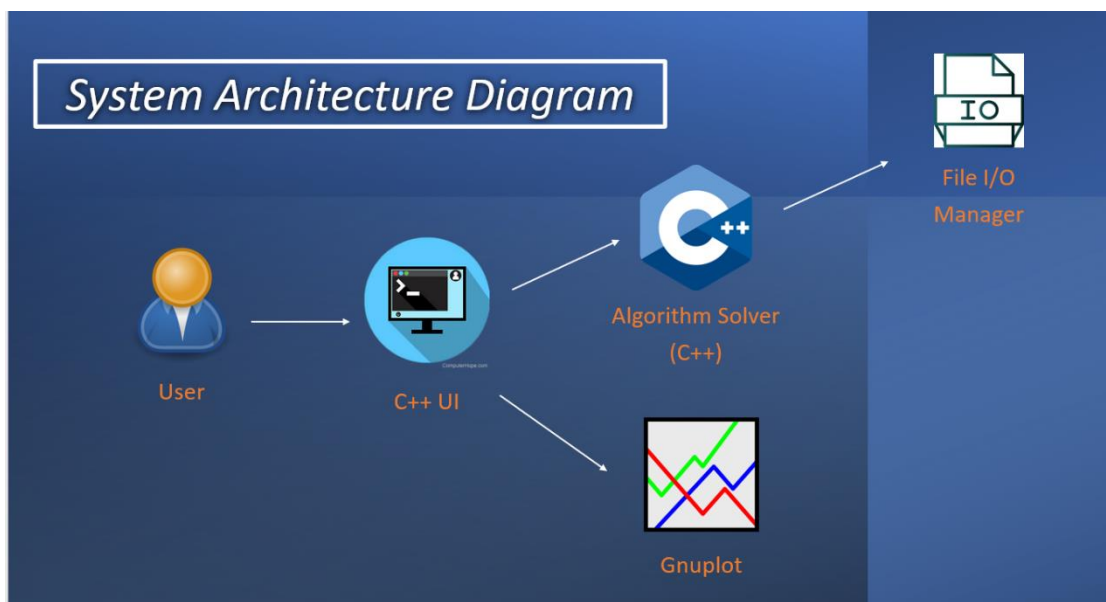


Figure 3.1.1: System architecture diagram.

The above figure illustrates the architectural framework of the system, defining the platform and environment integral to the entire process. Within this architecture, user interaction is facilitated through a C++ interface presented in the form of a command prompt. This interface

allows users to execute various functionalities, such as GQSO problem-solving and graph plotting.

The system's storage mechanism employs File I/O operations to efficiently store data in text files. Gnuplot plays a central role in the architecture, serving as the platform to visualize data by opening the Gnuplot's format file generated by the system. This integration ensures a user-friendly experience, allowing researchers to navigate the system effortlessly while utilizing powerful functionalities for GQSO analysis and visualization.

3.1.2 Use case diagram

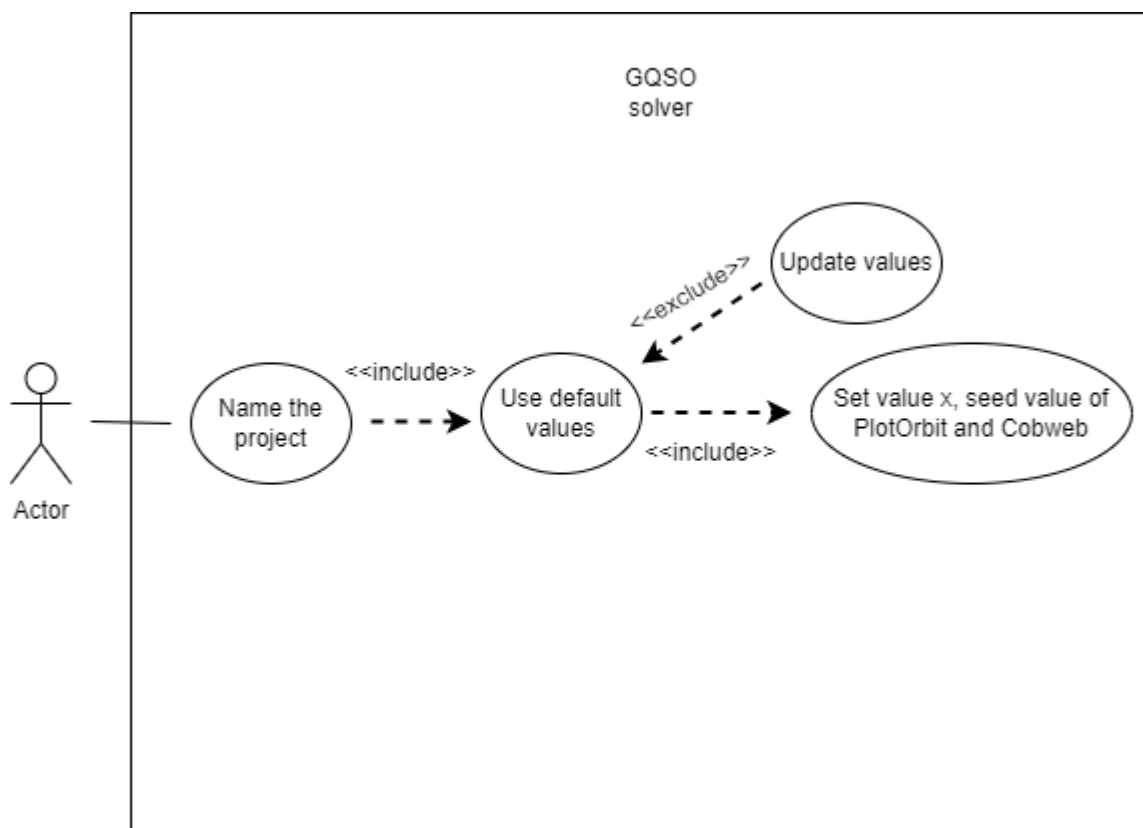


Figure 3.1.2: Use case diagram of the system.

This system is designed to make calculations lightning-fast, putting the user front and center in the action. It all begins with the user naming a project, a simple step that sets up a dedicated folder to neatly organize all the data that follows as well as preventing loss of data. After that, users can proceed into fine-tuning parameters like loop number, digits, and an array of coefficients tailored to specific cases. But even if users opt to stick with the defaults or new values, the system kindly lays out all the parameters for review to ensure all the values is satisfied before moving on. Then comes the part where users input values for 'x' and seed values

for PlotOrbit and Cobweb, ensuring all related values are obtained. Upon reaching the final stage, the system initiates its computational processes to generate the results. Once these calculations are concluded, the system automatically store the outcomes into the designated project folder.

3.1.3 Activity Diagram

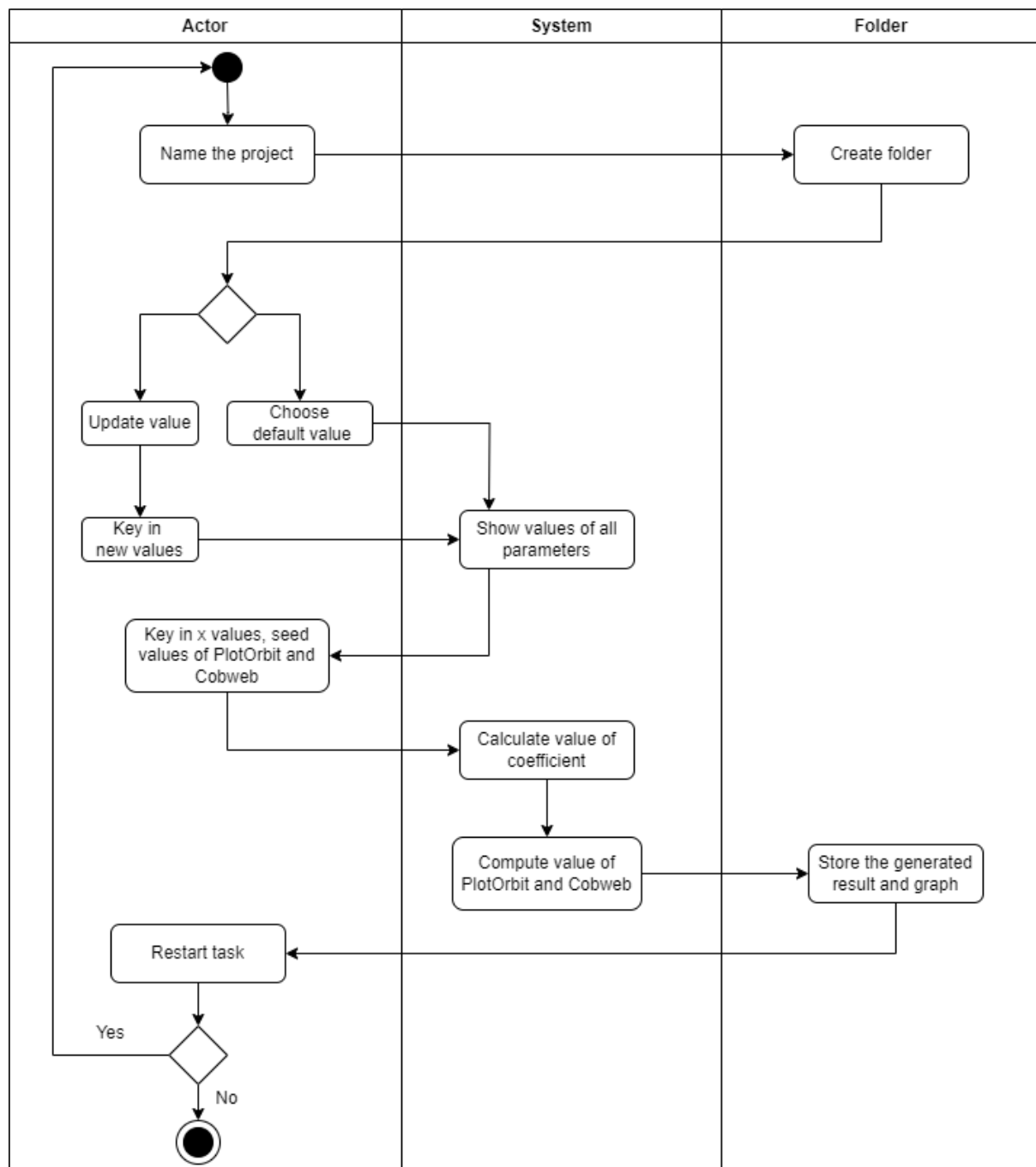


Figure 3.1.3: Activity diagram of system.

CHAPTER 3

The activity diagram offers valuable insight into the system's operations, facilitating a clearer understanding of its functionality. Within this system, three key components interact: the actor (user), the system itself, and the folder. Initially, the user engages with the system to name the project. Subsequently, the system autonomously generates a folder with the provided name. Following the creation of the folder, the system proceeds to a decision node, allowing users to opt between retaining default values or updating them. Should users choose to update, they advance to the input section to enter new values, which are subsequently displayed for confirmation. Alternatively, if users opt to stick with default values, these are also displayed. Following this, users input 'x' values along with seed values for PlotOrbit and Cobweb functionalities. With these inputs secured, the system commences computation of coefficients, utilizing them to further compute PlotOrbit and Cobweb. Without delay, the generated results and graphs are stored within the designated folder. At the end, users are prompted to decide whether to restart the process. Should they opt to do so, the system resets; otherwise, it will be ended.

3.2 Timeline

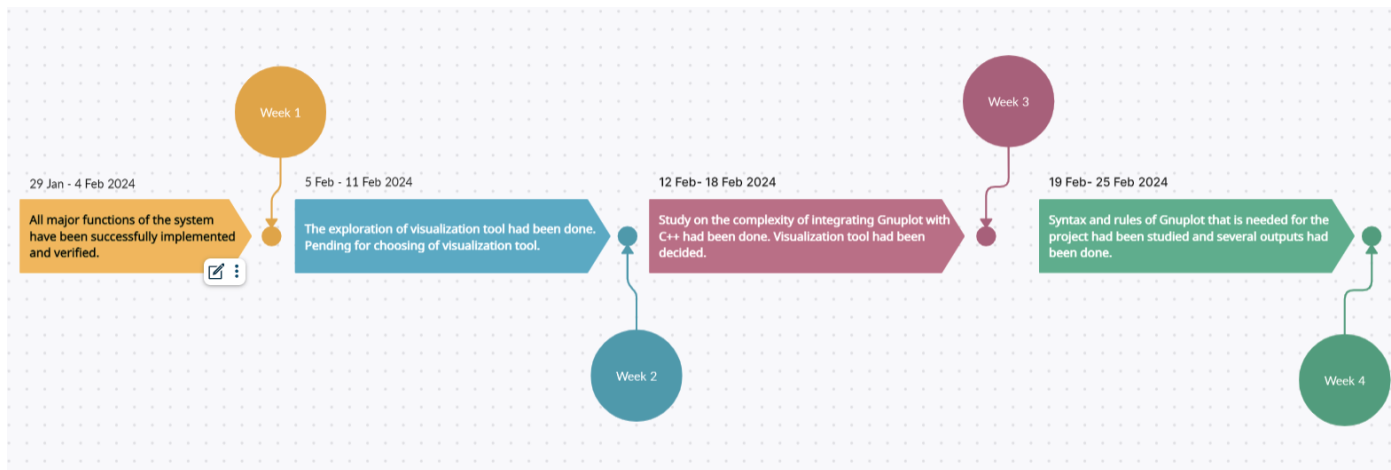


Figure 3.2.1: Timeline from Week 1 to Week 4.

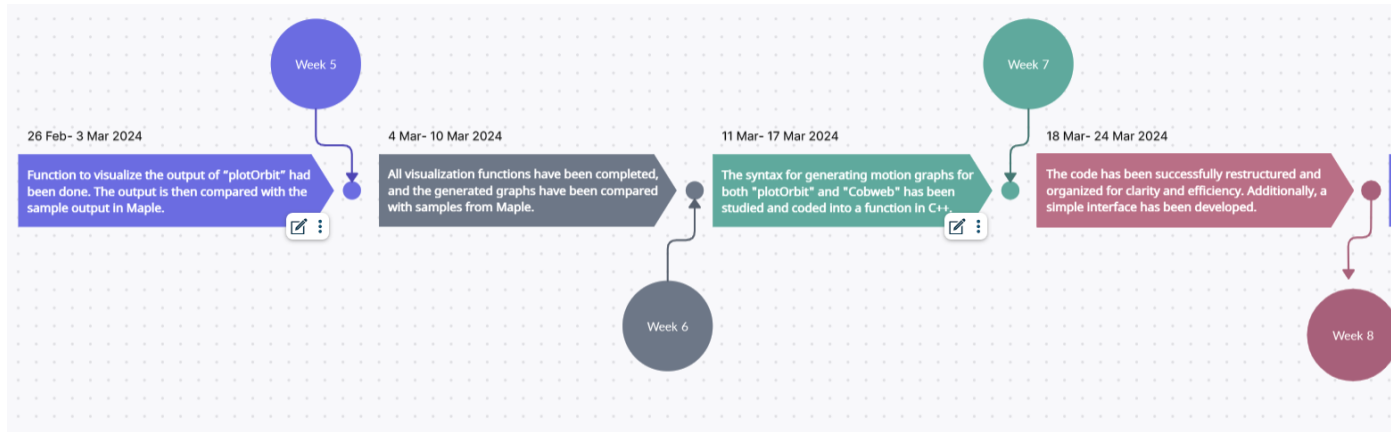


Figure 3.2.2: Timeline from Week 5 to Week 8.

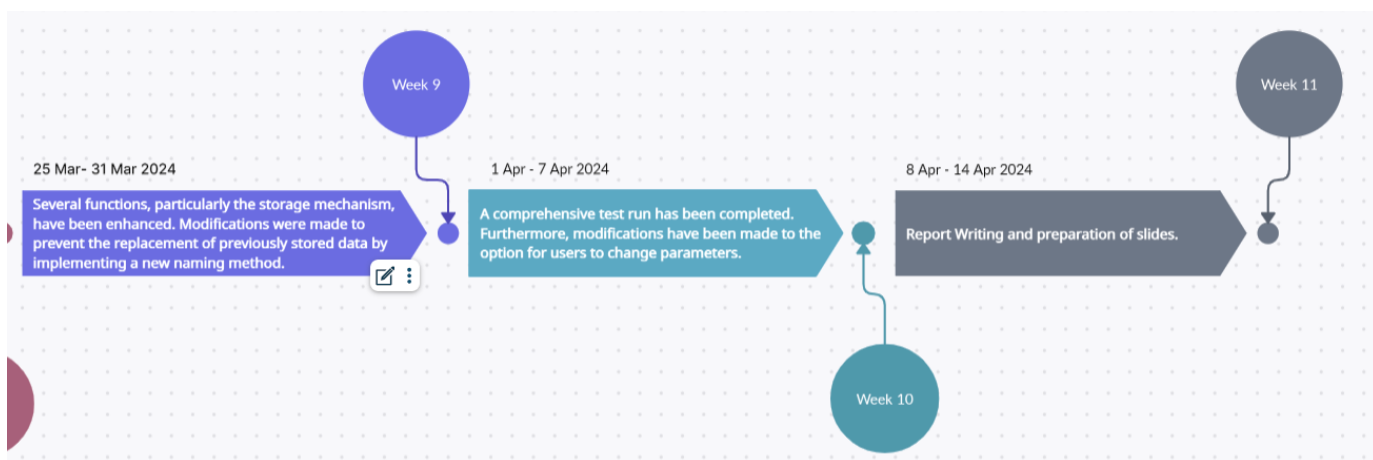


Figure 3.2.3: Timeline from Week 9 to Week 11.

The timeline provided demonstrates the progressive development of both the visualization function and the interface, with each stage focused on enhancing functionality and usability. Validation processes were employed throughout to ensure the final system met its objectives efficiently.

During the initial week, the primary functions were implemented, excluding the visualization aspect. Emphasis was placed on formatting each result generated by these functions, aiming to minimize potential bugs and easier the integration of the visualization function in subsequent stages.

Week 2 was dedicated to exploring different visualization tools. After careful consideration, both Matplotlib and Gnuplot were evaluated and compared. At the end, Gnuplot was selected as the preferred visualization tool due to its simpler implementation within the system compared to Matplotlib.

CHAPTER 3

During Week 3, efforts were focused on integrating Gnuplot into Visual Studio. However, due to the complexity of setting up the environment, it was determined that an external implementation of Gnuplot would be more feasible for ensuring user-friendliness within the system.

During Week 4, extensive study was conducted on the syntax and rules of Gnuplot to facilitate its integration into the system. Various outputs generated by Gnuplot were produced and compared against sample outputs from Maple to assess its reliability and consistency. This rigorous evaluation was crucial in ensuring the suitability of Gnuplot for the system's visualization needs.

During Week 5, with the reliability of Gnuplot validated, focus shifted to developing the function for visualizing results generated by PlotOrbit. The output produced by this function was compared with outputs from Maple to ensure accuracy and consistency. Additionally, efforts were made to implement the functionality for storing the generated graphs.

Week 6 marked the completion of visualizing results generated by the Cobweb function. These visualizations were thoroughly evaluated to ensure accuracy and effectiveness in conveying information. Additionally, the functionality for storing Cobweb graphs was implemented.

During Week 7, significant effort was dedicated to studying the syntax required to generate motion graphs using Gnuplot. Subsequently, functions for creating motion graphs for both PlotOrbit and Cobweb were developed. As these motion graphs were to be in .gif format and processed by Gnuplot rather than C++, attention was paid to optimizing the time taken for their generation.

During Week 8, significant efforts were dedicated to organizing and restructuring all code components for improved readability and maintainability, facilitating easier enhancements in subsequent weeks. Additionally, a simple interface was developed to enhance user experience, and initial test runs were conducted to identify and address any usability issues.

During Week 9, significant modifications were made to the storage mechanisms within the system. Specifically, a file naming mechanism was implemented during the initial stage, allowing users to specify unique names for their files. This ensured that all results and graphs generated would be stored separately, preventing any data from being overwritten during subsequent runs of the system. Additionally, efforts were made to obtain and display the execution time of the entire system at its conclusion. This allowed users to compare the

CHAPTER 3

efficiency between the C++ implementation and Maple, providing valuable insights into performance optimization opportunities.

During Week 10, a comprehensive test run of the system was conducted to identify any potential issues or areas for improvement. Based on the findings from this test run, modifications were made to the interface to enhance user flexibility and efficiency. Specifically, users were provided with the option to selectively update specific parameters rather than being required to update all values each time, thereby saving time and streamlining the user experience.

Chapter 4 System Design

4.1 System block diagram

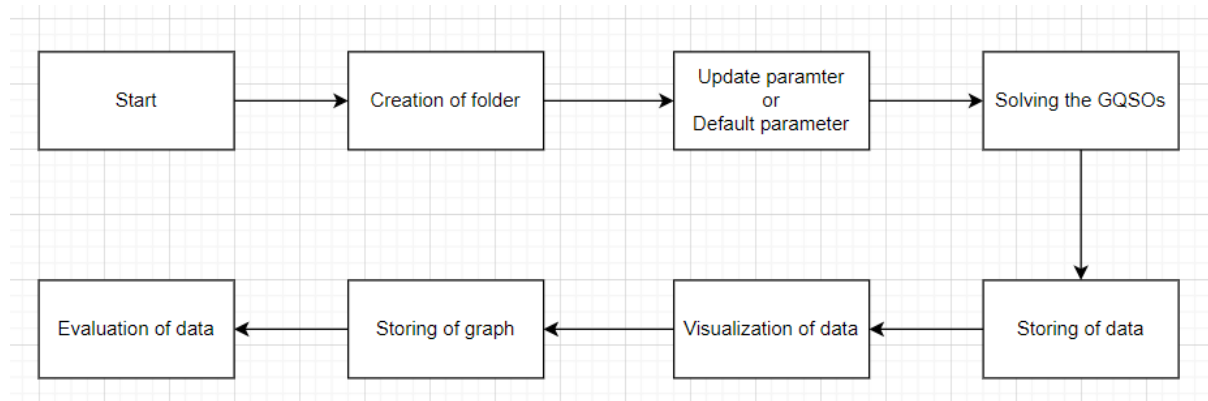


Figure 4.1: General system process design diagram

The above figure illustrates the procedural flow of the developed system. The system initialization begins with the creation of a folder, an essential step to establish a designated space for data storage. Users are prompted to name the folder, ensuring that data remains organized and avoiding potential overwrite issues during subsequent runs of the system. This preemptive measure safeguards against data loss or corruption, enhancing the overall reliability and stability of the system. The second step in this process involves the update of parameter or continue with default parameter, a crucial stage offering flexibility for real-world applications within specific research domains. Following the initialization of parameters to suit the requirements of the task, the next step involves the calling of functions as mentioned in previous sections that the employed methodology is procedural programming. The system initiates the algorithmic process by calling functions, addressing the GQSOs which are PlotOrbit and Cobweb using the user-input parameters. Upon computation of the results, the data is promptly stored in a text file and save to the folder created in first stage of system. Following the computation and storage phase, the system transitions to the data visualization stage. System will call the visualization function, triggering the generation of relevant graphs in Gnuplot's format file. Then, all the Gnuplot's format file will be save into the same file. At the end, accuracy and consistency of data will be evaluated.

4.2 System Components Specification

```
//Calculation(Part 1)
for (int k = p0; k <= p1; k++) {
    a00 = a00 + ((1 - r0) * pow(r0,k));
}
```

Figure 4.2.1: C++ code for solving A00 equation.

```
a1 = a00 - (2.0 * a01) + a02;
b1 = (2 * a01) - (2 * a02);
c1 = a02;
a2 = a10 - (2.0 * a11) + a12;
b2 = (2 * a11) - (2 * a12);
c2 = a12;
```

Figure 4.2.2: Calculation of coefficients.

Given that the equations involve summation, indicating the addition of a sequence of numbers, the code employs a For-loop to address the equation. The figure below illustrates a portion of the code responsible for solving the A00 equation. The same coding style will be maintained across all equations which are provided in previous.

Upon successful completion of the calculations, it becomes imperative to determine the quadratic equations and their coefficients as shown is Figure 4.2.2, as these values play a key role in the functionality of the "plotOrbit" and "CobWeb" functions. In addition, the $f(x)$ function will be constructed independently to prevent inconvenience when modifications are needed. The formula for the quadratic equation is expressed as follows:

```
double f_function(double x, double a1, double b1, double c1) {
    return a1 * x * x + b1 * x + c1; // Change this function as needed
}
```

Figure 4.2.3: The $f(x)$ function.

4.2.1 Function of PlotOrbit and Visualization of Plotorbit

```
auto expression = [&a1, &b1, &c1](double x) {
    return f_function(x, a1, b1, c1);
};
plotOrbit(expression,loop);
```

Figure 4.2.1.1: Call of PlotOrbit function.

```

void plotOrbit(function<double(double)> expression, int iterations) {
    vector<pair<int, double>> points;
    double seed;
    cout << "Enter the seed value for plotOrbit: ";
    cin >> seed;
    double z = seed;

```

Figure 4.2.1.2: PlotOrbit function.

```

    for (int i = 0; i <= iterations; i++) {
        for (int j = 0; j < i; j++) {
            // Update the seed by applying the function F
            z = expression(z);
        }

        // Store the point (i, F^i(seed))
        points.push_back(make_pair(i, z));
    }

```

Figure 4.2.1.3: Main calculation part of PlotOrbit.

Figure 4.1.2.4 illustrates the function of PlotOrbit, which relies on a quadratic function and a specified loop number. Thus, before PlotOrbit is called, as shown in Figure 4.1.2.3, the quadratic function is calculated. This involves the coefficients a_1 , a_2 , and a_3 which are calculated in a prior stage. Following the acquisition of all parameters, including the seed value, the system proceeds to the calculation phase shown in Figure 4.1.2.5.

```

ofstream pot_or_odd(folder + "/pot_or_odd.txt");
ofstream pot_or_even(folder + "/pot_or_even.txt");
for (const auto& point : points) {
    if (point.first % 2 == 0) {
        pot_or_even << point.first << " " << point.second << endl;
    }
    else {
        pot_or_odd << point.first << " " << point.second << endl;
    }
}
pot_or_odd.close();
pot_or_even.close();

Orbit();
Orbit_100();
Orbit_motion();

```

Figure 4.2.1.4: Storing and visualization of result of PlotOrbit.

CHAPTER 4

Upon completion of the calculation process, the system initiates the storage of results by referencing the data stored in the vector during the calculation phase. Text files are generated, and all results are written in the format "x y", where 'x' and 'y' represent the coordinates of the data points. Then, calling of visualization function will be done immediately.

```
void Orbit() {
    ofstream plot(folder + "/PlotOrbit.plt");
    plot << "set title \"Potor\"\n";
    plot << "set xlabel \"X-Axis\"\n";
    plot << "set ylabel \"Y-Axis\"\n";
    plot << "set style line 1 lc rgb '#FF0000' pt 7 ps 0.5\n";
    plot << "set style line 2 lc rgb '#61d493' pt 7 ps 0.5\n";
    plot << "plot \"potor_even.txt\" using 1:2 with points ls 1 title \"Even Iteration\"\n";
    plot << "replot \"potor_odd.txt\" using 1:2 with points ls 2 title \"Odd Iteration\"\n";

    plot.close();
}
```

Figure 4.2.1.5: Visualization of result of PlotOrbit.

```
void Orbit_100() {
    ofstream plot(folder + "/PlotOrbit_100.plt");
    plot << "set title \"Potor\"\n";
    plot << "set xlabel \"X-Axis\"\n";
    plot << "set ylabel \"Y-Axis\"\n";
    plot << "set style line 1 lc rgb '#FF0000' pt 7 ps 0.7\n";
    plot << "set style line 2 lc rgb '#61d493' pt 7 ps 0.7\n";
    plot << "plot \"potor_even.txt\" every ::0::49 using 1:2 with points ls 1 title \"Even Iteration\"\n";
    plot << "replot \"potor_odd.txt\" every ::0::49 using 1:2 with points ls 2 title \"Odd Iteration\"\n";

    plot.close();
}
```

Figure 4.2.1.6: Visualization of result of PlotOrbit for first 100 iterations.

```
void Orbit_motion() {
    ofstream gnuplotPipe(folder + "/plotOrbit_motion.plt");

    // Gnuplot commands
    gnuplotPipe << "set term gif animate delay 25\n";
    gnuplotPipe << "set output 'PlotOrbit_100_motion.gif'\n";
    gnuplotPipe << "set xlabel 'Iteration'\n";
    gnuplotPipe << "set ylabel 'Y'\n";
    gnuplotPipe << "set xrange [0:100]\n";
    gnuplotPipe << "set yrange [0:1]\n";
    gnuplotPipe << "set style line 1 lc rgb '#FF0000' pt 7 ps 0.5\n";
    gnuplotPipe << "set style line 2 lc rgb '#61d493' pt 7 ps 0.5\n";
    gnuplotPipe << "N = 70\n";
    gnuplotPipe << "do for [i=0:N] {\n";
    gnuplotPipe << "    plot 'potor_even.txt' every ::0::i using 1:2 with points ls 1 title sprintf('Even - Iteration %d', i*2), \\n";
    gnuplotPipe << "    'potor_odd.txt' every ::0::i using 1:2 with points ls 2 title sprintf('Odd - Iteration %d', 2*i+1)\n";
    gnuplotPipe << "}\n";
    gnuplotPipe << "unset output\n";

    gnuplotPipe.close();
}
```

Figure 4.2.1.7: Motion graph of PlotOrbit for first 100 iterations.

In Figures 4.2.1.5 through 4.2.1.7, the visualization function for PlotOrbit is presented. Since Gnuplot is implemented externally, these functions serve as output functions to generate files with the ".plt" format. These files are compatible with Gnuplot for visualization purposes. Furthermore, considering the potentially large number of iterations involved, an additional

visualization function is introduced as shown in Figure 4.2.1.6. This function specifically generates a graph using data from the first 100 iterations, ensuring clarity of the graph. Additionally, a motion graph is generated using a third visualization function outlined in Figure 4.2.1.7 for observation of data changes over each iteration. To optimize computational efficiency, only data from the first 100 iterations is utilized for plotting the motion graph instead of processing all data points.

4.2.2 Function of Cobweb and Visualization of Cobweb

```

void horizontal(double seed, double a1, double b1, double c1, int iterations) {
    ofstream horizontal_store(folder + "/horizontal.txt");

    vector < pair<pair<double, double>, pair<double, double>>> data;

    data.clear(); // Clear the vector to store new data
    for (int i = 0; i < iterations; i++) {
        double x1 = f_function(seed, a1, b1, c1);
        seed = x1;
        double x2 = f_function(seed, a1, b1, c1);
        pair<double, double> pair1 = make_pair(x1, x2);
        pair<double, double> pair2 = make_pair(x2, x2);

        data.push_back(make_pair(pair1, pair2));
    }
}

```

Figure 4.2.2.1: Calculation part of Cobweb (horizontal)

```

for (const auto& point : data) {
    if (check_x == round(point.first.second * 1e7) / 1e7 && check_y == round(point.first.first * 1e7) / 1e7) {
        if (num_divergence == 0) {
            horizontal_store << fixed << setprecision(7);
            horizontal_store << point.first.first << " " << point.first.second << " " << iter << endl;
            horizontal_store << point.second.first << " " << point.second.second << endl;
            num_divergence = iter;
        }
    }
    else {
        horizontal_store << fixed << setprecision(7);
        check_x = round(point.first.first * 1e7) / 1e7;
        check_y = round(point.first.second * 1e7) / 1e7;
        horizontal_store << point.first.first << " " << point.first.second << endl;
        horizontal_store << point.second.first << " " << point.second.second << endl;
    }
    iter++;
}
num_divergence = 0;
horizontal_store.close();

```

Figure 4.2.2.2: Storing result of Cobweb (horizontal).

```

void vertical(double seed, double a1, double b1, double c1, int iterations) {
    ofstream vertical_store(folder + "/vertical.txt");

    // Create a vector to store the line segments
    vector < pair<pair<double, double>, pair<double, double>> >data;

    data.clear(); // Clear the vector to store new data

    for (int j = 0; j <= iterations; j++) {
        double x1 = f_function(seed, a1, b1, c1);
        seed = x1;
        double x2 = f_function(seed, a1, b1, c1);

        pair<double, double> pair1 = make_pair(x1, x1);
        pair<double, double> pair2 = make_pair(x1, x2);

        data.push_back(make_pair(pair1, pair2));
    }
}

```

Figure 4.2.2.3: Calculation part of Cobweb (vertical)

```

for (const auto& segment : data) {
    if (check_x == round(segment.second.second * 1e7) / 1e7 && check_y == round(segment.second.first * 1e7) / 1e7) {
        if (num_divergence == 0) {
            vertical_store << fixed << setprecision(7);
            vertical_store << segment.first.first << " " << segment.first.second << " " << iter << endl;
            vertical_store << segment.second.first << " " << segment.second.second << endl;
            num_divergence = iter;
        }
    }
    else {
        vertical_store << fixed << setprecision(7);
        check_x = round(segment.second.first * 1e7) / 1e7;
        check_y = round(segment.second.second * 1e7) / 1e7;
        vertical_store << segment.first.first << " " << segment.first.second << endl;
        vertical_store << segment.second.first << " " << segment.second.second << endl;
    }
    iter++;
}

vertical_store.close();

```

Figure 4.2.2.4: Storing result of Cobweb (vertical).

The figures above show the horizontal and vertical aspect of Cobweb calculation. As depicted in Figure 4.2.2.1 and Figure 4.2.2.3, the necessary parameters for both functions include the seed value, coefficients, and number of iterations. Following this, calculations are executed, and values are stored into a vector, same as the process in PlotOrbit. Subsequently, upon completion of calculations, the system proceeds to store the data, outlined in Figure 4.2.2.2 and Figure 4.2.2.4. Here, a text file is created to document all results from the vector. Furthermore, the number of iterations required to achieve convergence is determined and recorded within the text file as well as inside a global variable for future referencing. This strategy aims to streamline computational time during the visualization of motion graph processing.

```

void cobweb() {
    ofstream cobweb(folder + "/cobweb.plt");

    cobweb << "plot \"graph.txt\" linecolor rgb \"red\",x linecolor rgb \"green\"\n";
    cobweb << "replot 'horizontal.txt' using 1:2 with linespoints lc rgb 'blue' title 'Horizontal'\n";
    cobweb << "replot 'vertical.txt' using 1:2 with linespoints lc rgb 'blue' title 'Vertical'\n";
    cobweb.close();
}

```

Figure 4.2.2.5: Visualization of Cobweb

```

void cobweb_motion() {
    ofstream gnuplotPipe(folder + "/cobweb_motion.plt");

    gnuplotPipe << "set term gif animate delay 10\n";
    gnuplotPipe << "set output 'cobweb_motion.gif'\n";
    gnuplotPipe << "set xlabel 'X'\n";
    gnuplotPipe << "set ylabel 'f(X)'\n";
    gnuplotPipe << "set xrange [0:1]\n";
    gnuplotPipe << "set yrange [0:1]\n";
    gnuplotPipe << "N = " << (num_divergence+3)*2 << "\n";
    gnuplotPipe << "do for [i=0:N] {\n";
    gnuplotPipe << "    plot 'horizontal.txt' every ::0::i using 1:2 with linespoints lc rgb 'blue' title sprintf('Horizontal - Iteration %d', i), \\n";
    gnuplotPipe << "    'vertical.txt' every ::0::i using 1:2 with linespoints lc rgb 'blue' title sprintf('Vertical - Iteration %d', i)\n";
    gnuplotPipe << "}\n";
    gnuplotPipe << "unset output\n";

    gnuplotPipe.close();
}

```

Figure 4.2.2.6: Motion graph of Cobweb.

```

void plotCOBWEB(double a1,double b1,double c1,int loop) {

    double seed;
    cout << "Enter the value of seed for cobweb: ";
    cin >> seed;
    horizontal(seed, a1, b1, c1, loop);
    vertical(seed, a1, b1, c1, loop);

    cobweb();
    cobweb_motion();
}

```

Figure 4.2.2.7: Function of Cobweb.

The approach for visualizing the Cobweb follows the same methodology as for PlotOrbit, where outputting files in Gnuplot's format is performed within these visualization functions. Given that all results are calculated and stored in text files within the same folder, graphs are plotted using these text files. Moreover, due to the high complexity of Cobweb, an optimization strategy is employed to save time constructing motion graphs with Gnuplot. The number of iterations required to achieve convergence, obtained from the global variable as mentioned previously, is utilized. This number of iterations serves as the threshold, ensuring that computational time is not wasted on repeating points after convergence. Lastly, Figure 4.2.2.7

CHAPTER 4

presents the full function for Cobweb, where the user is prompted to input the seed value. This function encompasses the entire process of Cobweb calculation and visualization, from parameter input to result generation.

Chapter 5 SYSTEM IMPLEMENTATION

5.1 Software and Hardware setup

For this project, a laptop computer serves as the hardware infrastructure. The software selection comprises essential tools, including Visual Studio 2019, Maple, and Gnuplot. Visual Studio 2019 is chosen for its renowned functionality and efficiency, providing the development environment for the required C++ programming. Maple plays a pivotal role, utilized for interpreting and analyzing pre-existing code addressing specific categories of GQSOs. Acting as a reference point, it enhances the speed of understanding the underlying concepts and structure of GQSOs during the development stage. Additionally, Gnuplot serves as the visualization tool, employed to create graphical representations of the final outcomes generated by the C++ code. These visualizations are crucial for evaluating the performance and results of the codes. The table below provides the specifications of the laptop. Before starting to develop the C++ code to address the GQSOs, there are two software needed to be installed and downloaded in my laptop:

1. Visual Studio 2019 Enterprise Edition 16.11.27
2. Gnuplot

Table 5.1 Specifications of laptop

Description	Specifications
Model	Predator Helios 300
Processor	Intel Core i5-10500H
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX 3060 Laptop GPU 6GB
Memory	16GB DDR4 RAM
Storage	1TB SATA SSD

5.2 System Operation

```

*****
**                Welcome to the                **
**                System                        **
*****

Give this project a name! : |

```

Figure 5.2.1: Creation of folder.

```

Give this project a name! : kk
Folder already exists. Please enter a different name.
Give this project a name! : |

```

Figure 5.2.2: Error handling of creation of folder

Figure 5.2.1 illustrates the initial step of the system, namely the naming section of the folder. This step is pivotal, as it establishes the folder where all results will be stored under the provided name. Subsequently, the system proceeds to verify the existence of the folder. If the folder already exists, a message indicating its existence is displayed, and the user is prompted to enter another name, as depicted in Figure 5.2.2.

```

*****
*                Created successfully                *
*****

*****
**                Welcome to the                **
**                Variable Update Menu          **
*****

Menu:
1. Update values
2. Stay with default values
Enter your choice: |

```

Figure 5.2.3: Update value or Default.

```

*****
Choose an option:
1. Update Loop number and Digit
2. Update r0,r1,r2 values
3. Update p0,p1,p2,p3 values
4. Done

*****
Enter your choice: 1

Loop number: 1500

Digit: 15|

```

Figure 5.2.4: Update values.

```

*****
**                Updated values:                **
*****

loop: 1500
digit: 15
p0: 0
p1: 0
p2: 1
p3: 10000
r0: 0.97499999999999997780
r1: 0.0389999999999999994
r2: 0.09750000000000000333

*****

```

Figure 5.2.5: Display updated values.

```

*****
**                Default values:                **
*****

loop: 1000
digit: 20
p0: 0
p1: 0
p2: 1
p3: 10000
r0: 0.97499999999999997780
r1: 0.0389999999999999994
r2: 0.09750000000000000333

*****

```

Figure 5.2.6: Display default values.

Figures 5.2.3 through 5.2.5 depict the process of updating parameter values within the system. Users have the option to update values, which redirects them to the updating value page. Here, users are prompted to choose which parameters to update. In Figure 5.2.4, the first option is selected, allowing users to update the value as desired. Subsequently, the system redirects back to the parameter option page, as shown in Figure 5.2.3, until the user selects the "Done" option. Upon completion, updated values are displayed, as illustrated in Figure 5.2.5. Additionally, if the user chooses to continue with default values, the system displays these default values on the screen to ensure user satisfaction with the parameters.

```
Enter x[0] value: 0.7
Calculation is done!

Enter the seed value for plotOrbit: 0
PlotOrbit is done!

Enter the value of seed for cobweb: 0.5
Cobweb is done!
```

Figure 5.2.7: PlotOrbit and Cobweb

```
Execution time: 196.65844329999998763014 seconds
Done! Do you want to restart? (Y/N): |
```

Figure 5.2.8: Restart of system

The figure above illustrates the initialization of PlotOrbit and Cobweb functions. As mentioned, both functions require a seed value before proceeding. Therefore, users are prompted to input the seed value at this point. Subsequently, the PlotOrbit and Cobweb functions are called, and a message is displayed once the calculations are completed. Additionally, before invoking these functions, users are asked to input "x" values to facilitate the calculation of coefficients required for subsequent functions. At this final stage, users have the option to either restart or end the system. If users opt to restart, the system redirects to the folder creation step, as depicted previously. Conversely, if users choose to end the system, the execution time is displayed for further analysis before concluding.

5.3 Implementation Issues and Challenges

Overall, the system implementation was designed with a focus on simplicity, considering the target audience of mathematicians or researchers without an IT background. However, a challenge arises during the decision-making process regarding the implementation of the visualization tool. Gnuplot was selected as the visualization tool, and the choice between internal and external implementation was considered. Through testing, it became evident that internally implementing Gnuplot in C++ proved more complex, as C++ offers fewer libraries compared to languages like Python. As a result, internal implementation would require more coding to achieve the desired output. Indeed, due to C++ has limited library support, leveraging powerful visualization tools like matplotlib, which is a Python library, would require considerable effort if chosen as the visualization approach for the system.

Given that Gnuplot is implemented externally, computing motion graphs may require additional computational time since it does not leverage C++'s computational advantages. However, the adoption of a time-saving technique, such as obtaining the number of iterations needed to achieve convergence, has significantly improved computational efficiency to an acceptable level. Nevertheless, if this number continues to increase, computational time may remain a concern.

5.4 Concluding Remarks

In conclusion, the system is intentionally designed to prioritize simplicity over a multitude of features or options that might confuse users. Its primary objective is to demonstrate the advantages of using C++ for solving problems involving a large number of calculations. Hence, users are allowed to modify parameters to illustrate real-world scenarios effectively. This streamlined approach ensures that users can focus on the core functionality of the system without being overwhelmed by unnecessary complexities, ultimately enhancing their understanding of C++'s computational capabilities. While the generation of motion graphs may raise concerns due to computational time, it is noteworthy that computing static graphs, even with a large number of iterations, remains extremely fast. As a result, motion graphs serve as an additional presentation of the results, providing a dynamic way to gain insights. While they may not play a crucial role in the system, they offer users an interactive and visual representation of the data, enhancing their understanding and analysis of the results.

Chapter 6 SYSTEM EVALUATION AND DISCUSSION

6.1 Testing Setup and Result

```

· Digits := 20 : M := 1000 :
· r0 := 0.975 : r1 :=  $\frac{1}{25}(r0)$  : r2 :=  $\frac{1}{10}(r0)$  :
· p0 := 0 : p1 := 0 :
· p2 := 1 : p3 := ∞ :
·  $A00 := \sum_{k=p0}^{p1} (1-r0) \cdot r0^k$  :  $A01 := \sum_{k=p0}^{p1} (1-r1) \cdot r1^k$  :  $A02 := \sum_{k=p0}^{p1} (1-r2) \cdot r2^k$  :
·  $A10 := \sum_{k=p2}^{p3} (1-r0) \cdot r0^k$  :  $A11 := \sum_{k=p2}^{p3} (1-r1) \cdot r1^k$  :  $A12 := \sum_{k=p2}^{p3} (1-r2) \cdot r2^k$  :
· A00, A01, A02
0.0250000000000000000000000000000000,  $\frac{24}{25}, \frac{9}{10}$ 
· A10, A11, A12
0.9750000000000000000000000000000000,  $\frac{1}{25}, \frac{1}{10}$ 
· a1 := A00 - 2·A01 + A02 : b1 := 2·A01 - 2·A02 : c1 := A02 :
· a2 := A10 - 2·A11 + A12 : b2 := 2·A11 - 2·A12 : c2 := A12 :
· a1, b1, c1
-0.9950000000000000000000000000000000,  $\frac{3}{25}, \frac{9}{10}$ 
· a2, b2, c2
0.9950000000000000000000000000000000,  $-\frac{3}{25}, \frac{1}{10}$ 
· x[0] := 0.75 :
· y[0] := 1 - x[0] :

```

Figure 6.1.1: Parameter in Maple.

```

cobweb(f(x), x = 0 ..1, x = 0.7, M);

plotorbit(f(x), x = 0, M);

```

Figure 6.1.2: Seed value of PlotOrbit and Cobweb.

```

*****
**              Default values:              **
*****

loop: 1000
digit: 20
p0: 0
p1: 0
p2: 1
p3: 10000
r0: 0.97499999999999997780
r1: 0.0389999999999999994
r2: 0.09750000000000000033

*****

Enter x[0] value: 0.75
Calculation is done!

Enter the seed value for plotOrbit: 0
PlotOrbit is done!

Enter the value of seed for cobweb: 0.7
Cobweb is done!

```

Figure 6.1.3: Parameter in C++ system.

The figures above depict the parameters and seed values utilized for the same case in both Maple and the C++ system. This step is essential to ensure that both systems are solving the same case under identical conditions, facilitating accurate comparison and validation of results. Furthermore, it is important to note that the values may exhibit minor differences due to variations in the decimal places used. C++ allows for more decimal places, resulting in calculations that may offer greater precision compared to results obtained from Maple. These differences in precision should be taken into consideration when comparing the outputs from both systems.

x[1]=0.430312, y[1]=0.569688	x[1]: 0.430844	y[1]: 0.569156
x[2]=0.767394, y[2]=0.232606	x[2]: 0.768303	y[2]: 0.231697
x[3]=0.406137, y[3]=0.593863	x[3]: 0.405348	y[3]: 0.594652
x[4]=0.784614, y[4]=0.215386	x[4]: 0.786522	y[4]: 0.213478
x[50]=0.862803, y[50]=0.137197	x[50]: 0.867335	y[50]: 0.132665
x[51]=0.26283, y[51]=0.73717	x[51]: 0.255845	y[51]: 0.744155
x[52]=0.862806, y[52]=0.137194	x[52]: 0.867337	y[52]: 0.132663
x[53]=0.262825, y[53]=0.737175	x[53]: 0.255842	y[53]: 0.744158
x[100]=0.86281, y[100]=0.13719	x[100]: 0.867340	y[100]: 0.132660
x[101]=0.262818, y[101]=0.737182	x[101]: 0.255837	y[101]: 0.744163
x[102]=0.86281, y[102]=0.13719	x[102]: 0.867340	y[102]: 0.132660
x[103]=0.262818, y[103]=0.737182	x[103]: 0.255837	y[103]: 0.744163

Figure 6.1.4: Comparison of Result

C++ demonstrates exceptional computational efficiency compared to Maple, with the former generating results swiftly, even when the number of iterations is relatively small. In contrast, Maple exhibits slow computational speeds, taking more than 30 seconds to compute results for the same problem. This performance difference highlights C++'s capability to handle GQSOs with remarkable speed. However, Figure 4.3.1 reveals a slight discrepancy between preliminary results produced by C++ (on the right-hand side) and Maple (on the left-hand side). The minor differences in values may lead to slight variations in future graph plotting, posing a challenge that needs to be addressed in subsequent work. The same problem should be appeared on “plotOrbit” and “CobWeb” function as the coefficient had been affected by the slightly discrepancy. This remains an assumption as the sample Maple code never display the coordinates explicitly, requiring additional time and effort to accurately pinpoint the exact differences between the results generated by Maple and C++.

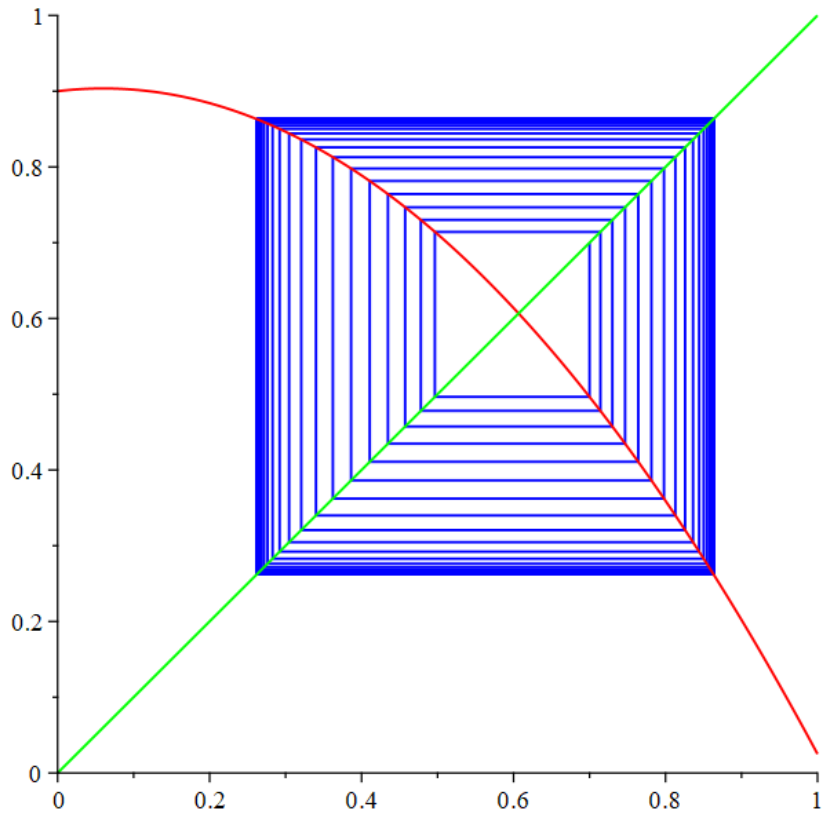


Figure 6.1.5: Cobweb from Maple.

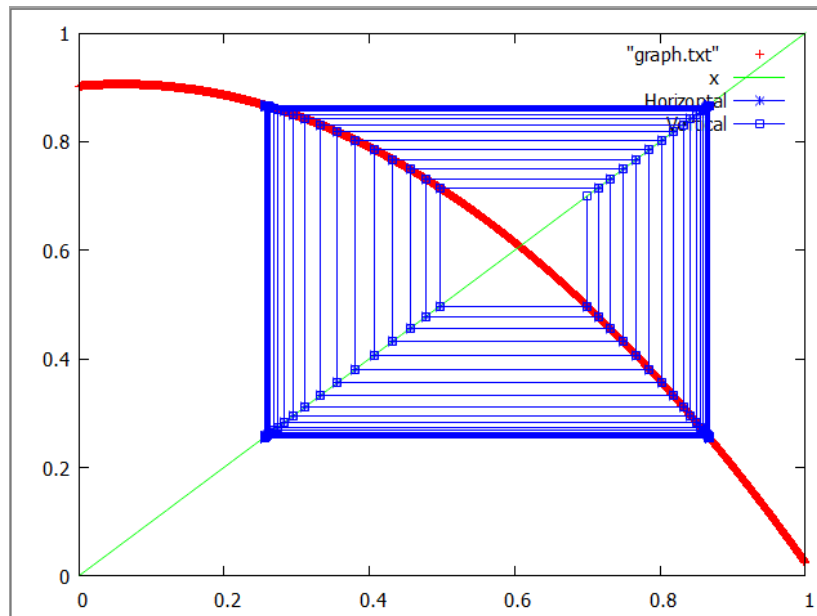


Figure 6.1.6: Cobweb from Gnuplot.

CHAPTER 6

In Figures 6.1.5 and 6.1.6, the Cobweb plots from Maple and Gnuplot, respectively, are depicted. A minor difference is observed at the starting point of the graph between Maple and Gnuplot. However, as previously mentioned, this discrepancy may not be a cause for concern, as it can be attributed to the slight differences in decimal places between the two systems. Overall, these minor variations are expected and do not significantly impact the interpretation of the results. Additionally, it is noteworthy that apart from the minor differences at the starting point, the graph generated from Gnuplot closely resembles the graph from Maple. This consistency indicates that the accuracy and reliability of the system are sufficient, providing confidence in the validity of the results obtained from the C++ implementation.

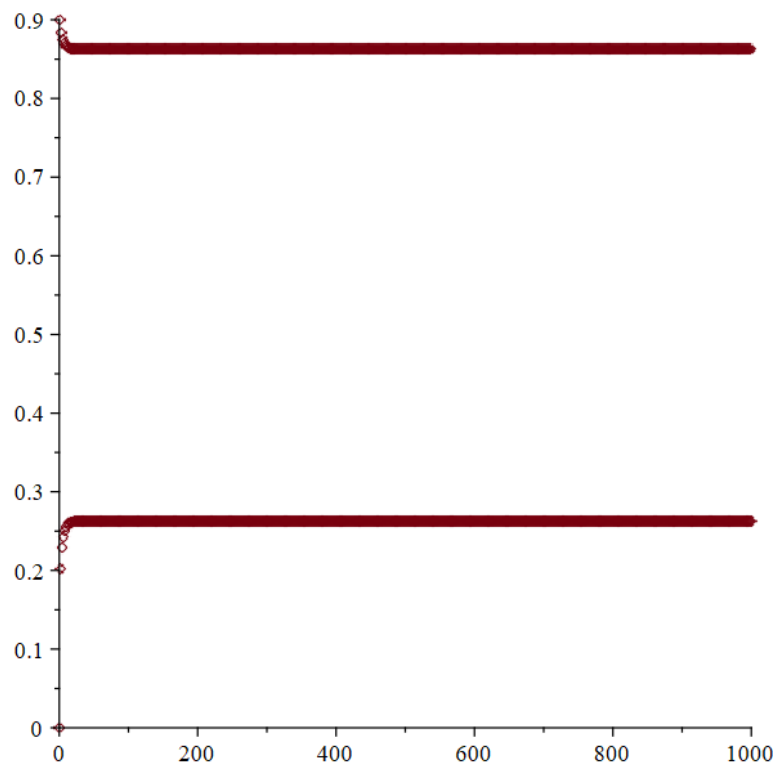


Figure 6.1.7: PlotOrbit from Maple.

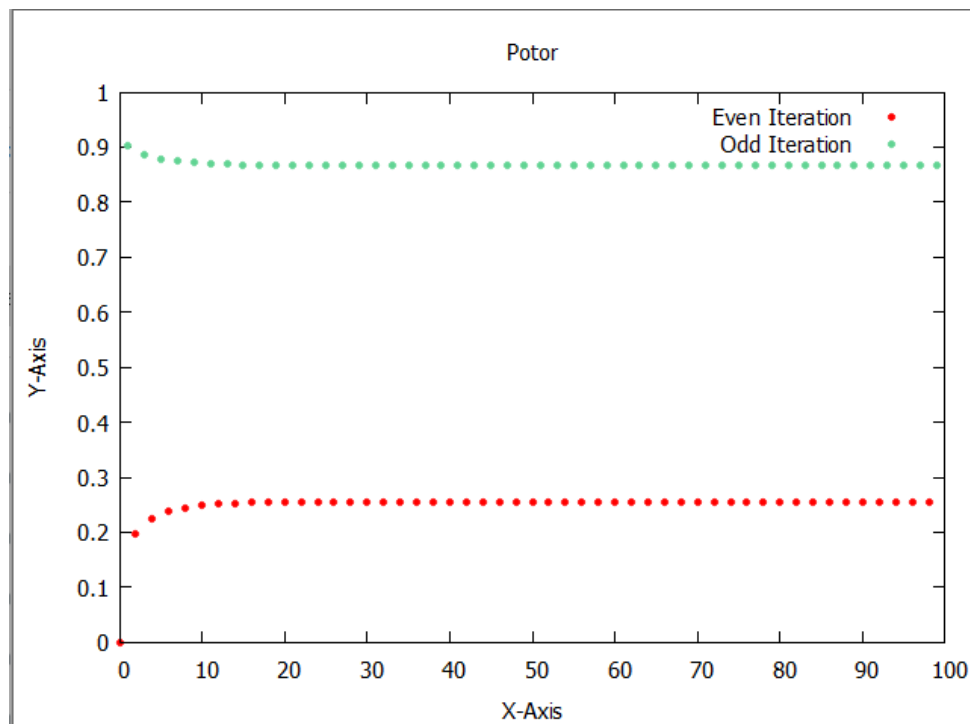


Figure 6.1.8: PlotOrbit from Gnuplot.

Time: 28.10s

Execution time: 8.53222710000000006403 seconds

Figure 6.1.9: Comparison of execution time.

When analyzing the PlotOrbit function, it shows that the graph generated from Gnuplot closely resembles that from Maple. However, Gnuplot offers a distinct advantage over Maple by allowing the even and odd iterations to be displayed separately with ease. This feature provides valuable insight into the results, enhancing the interpretability of the graph.

Furthermore, the execution times displayed in Figure 6.1.9 highlight a significant disparity between Maple and C++. Maple exhibits a longer execution time of 28.10 seconds compared to C++, which completes its processes in 8.53 seconds. It is worth noting that the 8.53 seconds in C++ encompasses all processes, including folder creation, parameter value determination, coefficient calculation, PlotOrbit and Cobweb computation, data, and graph storage whereas Maple takes 28.10 seconds for only solving PlotOrbit and Cobweb. C++ demonstrates remarkable efficiency in the solving part compared to Maple, completing computations significantly faster. Additionally, C++ is capable performing other useful features such as folder creation, parameter value determination, data storage, and graph generation within the

same timeframe. This showcases the capabilities and computational power of C++, making it a preferred choice for tackling computationally intensive tasks.

6.2 Project Challenges

Indeed, encountering challenges with understanding Maple syntax is understandable given its less common usage compared to other programming languages. Developing a reliable C++ system to replicate the functionalities of Maple requires additional effort in studying the syntax and properties of Maple programming. Moreover, visualization of results poses another set of challenges, particularly in ensuring accuracy and consistency between the outputs generated by Maple and C++. Overcoming these challenges may involve thorough research and experimentation. In addition, the development of motion graphs adds a dynamic dimension to the presentation of data. Additionally, mastering the commands and capabilities of Gnuplot is essential to ensure the smooth generation of motion graphs while minimizing computational time. Unlike static graphs, motion graphs require additional computational resources to plot trajectories over time. Since Gnuplot does not inherently leverage computational advantages from C++, it is important to optimize the motion graph generation process within Gnuplot itself. This includes implementing efficient plotting techniques, optimizing data processing, and leveraging Gnuplot's features to streamline the visualization process.

6.3 Objectives Evaluation

In Chapter 1, we outlined our objectives: to create a system capable of handling extensive calculations, visualizing data effectively, and providing robust storage mechanisms. It is pleased to report that the system has been successfully developed and has met all the proposed objectives with outstanding performance. In addition, an exciting extra feature has been developed: the motion graph. Recognizing the dynamic nature of the project, which revolves around changes over iterations, the motion graph emerges as a fitting approach to illustrate the data. By incorporating this feature, we elevate the visualization capabilities of the system, allowing for a more nuanced understanding of the iterative processes at play. While the motion graph enhances our system's visual appeal, it is essential to consider its impact on computational efficiency. Despite implementing techniques to streamline processes, using a vast number of iterations might strain efficiency. This could hinder our objective of efficiently visualizing results. As we refine our system, finding a balance between visualization depth and computational efficiency will be key of future.

6.4 Concluding Remarks

In conclusion, the developed system surpasses Maple in performance under similar conditions and cases. Its user-friendly interface caters to individuals with limited IT knowledge, ensuring accessibility and ease of use. Moreover, the system incorporates a robust storage mechanism, safeguarding against data loss that could impede future analysis processes. Additionally, the execution time of our developed system is significantly faster compared to Maple. This notable speed advantage can greatly benefit researchers and mathematicians engaged in studying such operators, enhancing their efficiency and productivity. By reducing the time required for calculations and analysis, the system empowers users to delve deeper into their research and make strides in their respective fields at an accelerated pace. Furthermore, the system offers both static and motion-based visualization approaches, providing users with a comprehensive toolkit for interpreting results. This dual visualization capability enhances the depth of insight gained from the data, facilitating further analysis and exploration. Whether examining static snapshots or dynamic trends over iterations, users can gain a richer understanding of their results, empowering them to make informed decisions and discoveries.

Chapter 7 CONCLUSION AND RECOMMENDATION

7.1 Conclusion

As a conclusion, the project has been successfully developed, demonstrating outstanding performance compared to Maple programming. This success signifies the fulfillment of our proposed objectives. Additionally, the integration of an extra feature, a dynamic approach to visualizing data, further enhances the system's functionality. However, it is important to acknowledge the challenge posed by the potential increase in computation time when dealing with a large number of iterations, particularly in generating motion graphs. This challenge highlights the need for ongoing optimization efforts to ensure efficient performance, maintaining a balance between functionality and computational resources.

7.2 Recommendation

To enhance the reliability of the developed system, it is imperative to expand the testing process beyond simple cases. Incorporating a diverse range of quadratic equations, reflective of real-world scenarios, will provide a more robust assessment of the system's capabilities. By exposing the system to a variety of challenges and complexities inherent in practical applications, it can ensure its reliability and effectiveness across a broader spectrum of scenarios. This approach will not only validate the system's performance but also bolster confidence in its utility for real-world practice.

REFERENCES

- [1] S. N. Bernstein, "Solution of a Mathematical Problem Connected with the Theory of Heredity," *The Annals of Mathematical Statistics*, vol. 13, no. 1, pp. 53-61, 1942.
- [2] F. Mukhamedov, I. Qaralleh, and W. N. F. A. W. Rozali, "On ξ_a -Quadratic Stochastic Operators on 2-D Simplex," *Sains Malaysiana*, vol. 43, no. 8, pp. 1275-1281, 2013.
- [3] K. Ftameh, C. H. Pah, "On Three-Dimensional Mixing Geometric Quadratic Stochastic Operators," *Mathematics and Statistics*, Vol. 9, No. 2, pp. 151 - 158, 2021. DOI: 10.13189/ms.2021.090209.
- [4] M Paterno et al 2014 *J. Phys.: Conf. Ser.* 513 052026.
- [5] Mukhamedov, F. and Embong, A. F., "Extremity of b-bistochastic Quadratic Stochastic Operators on 2D Simplex," *Malaysian Journal of Mathematical Sciences*, vol. 11, no. 2, pp. 119-139, 2017.
- [6] Zahedi et al 2021 *J. Phys.: Conf. Ser.* 1898 01203.
- [7] D. Rassokhin, "The C++ programming language in cheminformatics and computational chemistry," *Journal of Cheminformatics*, vol. 12, pp. 10, 2020. doi: 10.1186/s13321-020-0415-y.
- [8] J. Hunter, D. Dale, E. Firing, and M. Droettboom, "Matplotlib Release 2.0.2," 2017. Available: <https://matplotlib.org/2.0.2/Matplotlib.pdf>.

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 1
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

All major functions of the system have been successfully implemented and verified, ensuring their proper operation and output format. The only pending aspect is the visualization function, which is slated for completion soon.

2. WORK TO BE DONE

Next week, I will commence the exploration of appropriate visualization tools to enhance our system. If multiple of tools have been selected, then comparison will be the one of the tasks as well.

3. PROBLEMS ENCOUNTERED

At this early stage, I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

Progress may experience delays as the validation of previous work is taking longer than anticipated.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 2
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

The exploration of visualization tool had been done. Since multiple tools are selected so comparison between potential tools had been done as well. The final decision on the visualization tool had been made which is Gnuplot.

2. WORK TO BE DONE

In next week, I will conduct a study on the complexity of integrating Gnuplot with C++ to determine the most efficient approach. This evaluation will help us decide whether to set up Gnuplot directly within our C++ codebase or utilize it externally.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

The delayed progress has been successfully mitigated by allocating additional time and effort to the project.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 3
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

Study on the complexity of integrating Gnuplot with C++ had been done. I had decided to implement Gnuplot externally due to high complexity during the set-up process.

2. WORK TO BE DONE

Next week, I will thoroughly explore the syntax, rules, and regulations of Gnuplot to better understand its capabilities for better implementation in the system.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

Although task started later than planned due to personal reasons, I have managed to catch up on progress.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 4
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

Syntax and rules of Gnuplot that is needed for the project had been studied and several outputs had been done. The sample outputs are compared with the sample output in Maple.

2. WORK TO BE DONE

Next week, I will begin implementing Gnuplot in C++. Our focus will be on coding functions to generate files containing Gnuplot's syntax, laying the groundwork for integrating visualization into our project.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

All the tasks are conducted on time. No delay of progress.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 5
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

Function to visualize the output of “plotOrbit” had been done. The output is then compared with the sample output in Maple.

2. WORK TO BE DONE

Next week, I will allocate more effort towards visualizing the data from the Cobweb function. Additionally, I will conduct a comparison between Gnuplot and Maple to ensure the reliability of our visualization approach.

3. PROBLEMS ENCOUNTERED

The visualization of the Cobweb function could not be completed this week due to its high complexity.

4. SELF EVALUATION OF THE PROGRESS

While the visualization of the output of the Cobweb function couldn't be completed this week, there's still enough time to accommodate a slight delay for it.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 6
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

All visualization functions have been completed, and the generated graphs have been compared with samples from Maple.

2. WORK TO BE DONE

In next week, effort will be put on how to create a motion graph for function “plotOrbit” and “Cobweb”. Revise the syntax of Gnuplot will be conducted as previous output involve only static graph.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

The task that was unable to finish on time last week has been successfully completed this week, without causing any delays to the tasks scheduled for this week. Overall, the progress is good.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 7
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

The syntax for generating motion graphs for both "plotOrbit" and "Cobweb" has been studied and coded into a function in C++.

2. WORK TO BE DONE

Next week, I will focus on restructuring the code and initiating the design of a simple interface.

3. PROBLEMS ENCOUNTERED

Generating the motion graph may take some time, as it is done using Gnuplot rather than C++, which means there is no computational advantage.

4. SELF EVALUATION OF THE PROGRESS

Overall, the progress is good.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 8
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

The code has been successfully restructured and organized for clarity and efficiency. Additionally, a simple interface has been developed and thoroughly validated to ensure smooth interaction for users.

2. WORK TO BE DONE

Next week, I will conduct a comprehensive review of all previous work to ensure the functionality of the system. This process will also help identify and address any bugs, leading to further improvements in the system.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

Overall, the progress is good.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 9
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

Several functions, particularly the storage mechanism, have been enhanced. Modifications were made to prevent the replacement of previously stored data by implementing a new naming method. Additionally, a feature displaying the execution time for functions involving computation has been added to facilitate performance comparison between C++ and Maple systems.

2. WORK TO BE DONE

Next week, a full test run will be done to validate whether all the function work nicely and ensuring the smoothness of the system flow.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS

Overall, the progress is good.

Siti Nurlaili

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	Study week no.: 10
Student Name & ID: LIM YIK HENG 2003395	
Supervisor: Dr. Siti Nurlaili Karim	
Project Title: SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++	

1. WORK DONE

A comprehensive test run has been completed, ensuring thorough evaluation of the system's functionality. Furthermore, modifications have been made to the option for users to change parameters. Now, users can select specific sets of variables they wish to modify, rather than having to adjust all variables, thereby saving time, and enhancing usability.

2. WORK TO BE DONE

Now that all development tasks have been completed, the focus will shift towards preparing the report and presentation in the coming weeks.

3. PROBLEMS ENCOUNTERED

I am pleased to report that no critical problems have arisen.

4. SELF EVALUATION OF THE PROGRESS


Overall, the progress is good.

Siti Nurlaili

Supervisor's signature



Student's signature



FACULTY OF INFORMATION
COMMUNICATION AND TECHNOLOGY

BREAKING NEWS


SOLVING GEOMETRIC QUADRATIC STOCHASTIC OPERATORS USING C++

INTRODUCTION

GQSOs is not a trend due to its complexity. Maple could solve it but challenges were posed when solving GQSOs such as time consuming, lack of user-friendly interface as well as requiring a strong IT background.

OBJECTIVE

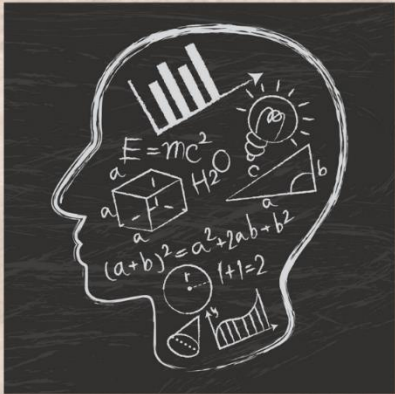
To deliver a C++ program which consists of user-friendly interface, outstanding computational time and low requirement of IT background.



PROPOSED METHOD

Procedural Programming

“Divided into a series of smaller, self-contained procedures or blocks of code.”



WHY C++ BUT NOT OTHERS?

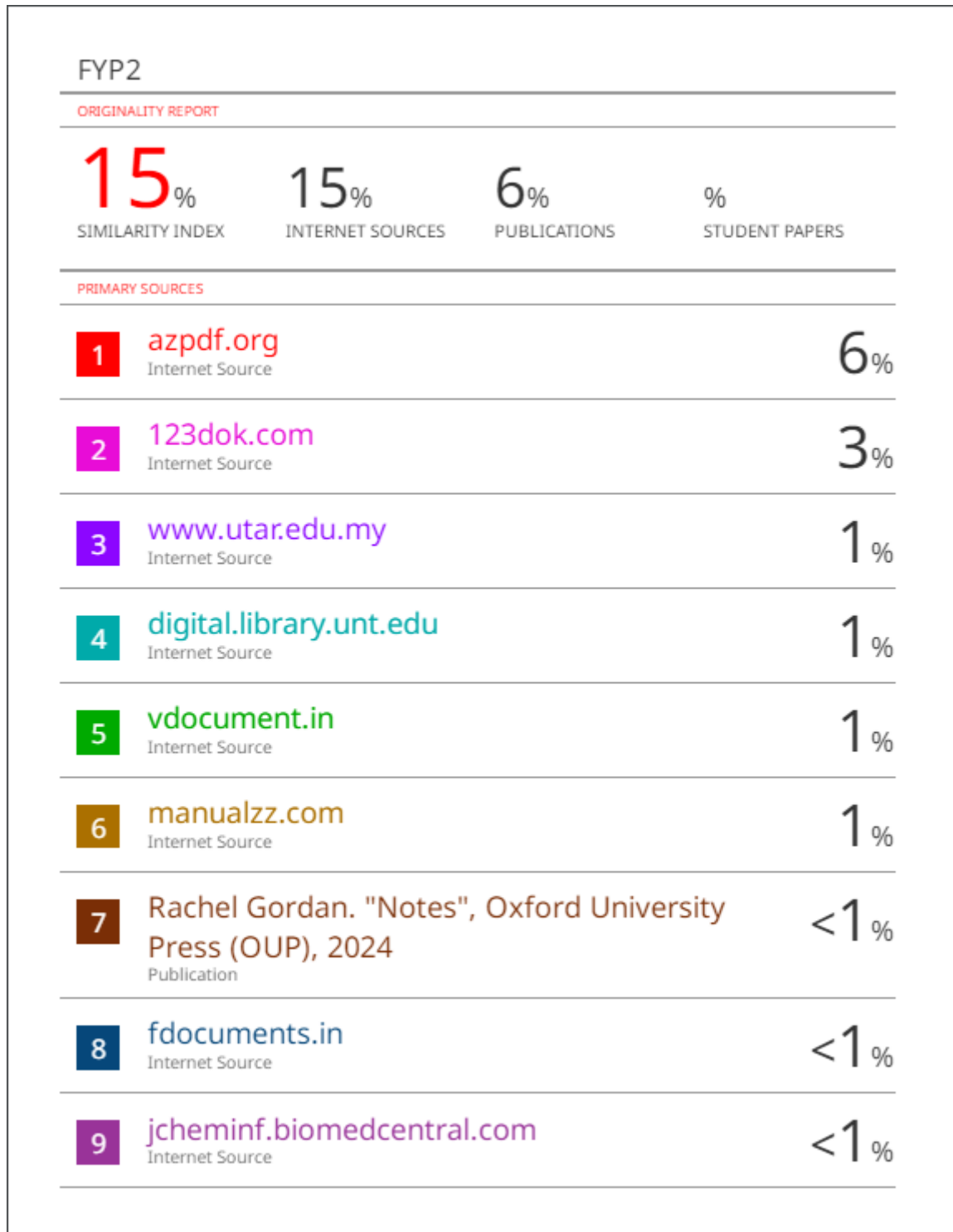
- C++ is designed for heavy computational task and easier to create portable code.
- Maple need solid background in programming but C++ does not.

CONCLUSION

Maple may be good but C++ is better in terms computational power as well as storage mechanism.

Project developer: Lim Yik Heng
Project supervisor: Dr. Siti Nurlaili Karim

PLAGIARISM CHECK RESULT



PLAGIARISM CHECK RESULT

10	www.coursehero.com Internet Source	<1 %
11	kipdf.com Internet Source	<1 %
12	Ton Duc Thang University Publication	<1 %
13	hdl.handle.net Internet Source	<1 %
14	me.emu.edu.tr Internet Source	<1 %
15	dspace.library.uvic.ca:8080 Internet Source	<1 %
16	eprints.utas.edu.au Internet Source	<1 %
17	perakhainan8.gbs2u.com Internet Source	<1 %
18	Humphreys, Rosalind K., Ruxton, Graeme D.. "Presenting Scientific Data in R", Presenting Scientific Data in R, 2022 Publication	<1 %
19	Zahedi, Marwan Affandi, Suparni, Yenny Suzana, Fachrur Razi. "Maple as a tool to understand marketing game for up to four companies", Journal of Physics: Conference Series, 2021 Publication	<1 %

20 A. Alsarayreh, I. Qaralleh, M. Z. Ahmad. " $\xi^{\{as\}}$ -QUADRATIC STOCHASTIC OPERATORS IN TWO-DIMENSIONAL SIMPLEX AND THEIR BEHAVIOR", JP Journal of Algebra, Number Theory and Applications, 2017 <1%
Publication

21 Jianping Wang, Xueyan Zhang, Guohong Gao, Yingying Lv, Qian Li, Zhiyu Li, Chengchao Wang, Guanglan Chen. "Open Pose Mask R-CNN network for Individual Cattle Recognition", IEEE Access, 2023 <1%
Publication

22 João Eduardo Gentil Lé. "Melhorias e ampliação da Banda W do sistema de caracterização de antenas do EPUSP Centro mmW .", Universidade de São Paulo. Agência de Bibliotecas e Coleções Digitais, 2023 <1%
Publication

23 Larry A. Glasgow. "Applied Mathematics for Science and Engineering", Wiley, 2014 <1%
Publication

24 Maksym Gaiduk, Ángel Serrano Alarcón, Ralf Seepold, Natividad Martínez Madrid. "Conception of a home-based sleep apnoea identification and monitoring system", Procedia Computer Science, 2023 <1%
Publication

PLAGIARISM CHECK RESULT

25 Mohammed K. Elghoul, Sayed F. Bahgat, Ashraf S. Hussein, Safwat H. Hamad. "Management of medical record data with multi-level security on Amazon Web Services", SN Applied Sciences, 2023
Publication

<1 %

26 dspace.daffodilvarsity.edu.bd:8080
Internet Source

<1 %

27 dspace.lboro.ac.uk
Internet Source

<1 %

28 research.brighton.ac.uk
Internet Source

<1 %

29 www.acegadgets.com.my
Internet Source

<1 %

Exclude quotes Off

Exclude matches Off

Exclude bibliography On

ORIGINALITY REPORT

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Lim Yik Heng
ID Number(s)	20ACB03395
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Solving Geometric Quadratic Stochastic Operators using C++

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u>15</u> % Similarity by source Internet Sources: <u>15</u> % Publications: <u>6</u> % Student Papers: <u>0</u> %	The similarity index does not exceed the limit.
Number of individual sources listed of more than 3% similarity: <u>1</u>	The similarity context is not significant.
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Siti Nurlaili

Signature of Supervisor

Name: Dr. Siti Nurlaili Karim

Date: 23/4/2024

Signature of Co-Supervisor

Name: _____

Date: _____

CHECKLIST



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB03395
Student Name	Lim Yik Heng
Supervisor Name	Dr. Siti Nurlaili Karim

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✓	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 22/4/2024