COMPUTER PROGRAMMING APPROACH TO HYDRAULIC FLOW BALANCING IN LOOPED NETWORKS USING HARDY CROSS METHOD

BY

MOSES, George Odion

ENG2006193

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF ENGINEERING(B.Eng.) DEGREE.

IN

THE DEPARTMENT OF CIVIL ENGINEERING,

FACULTY OF ENGINEERING,

UNIVERSITY OF BENIN, BENIN CITY, NIGERIA.

NOVEMBER, 2025.

PLAGIARISM

This work **COMPUTER PROGRAMMING APPROACH TO HYDRAULIC FLOW BALANCING IN LOOPED NETWORKS USING HARDY CROSS METHOD** by MOSES, George Odion with Matriculation Number, ENG2006193 of the department of Civil Engineering, Faculty of Engineering, University of Benin City, Edo State, Nigeria, has PASSED the PLAGIARISM TEST.

PROJECT COORDINATOR:

Name: ENGR. EHI ORIA-USIFO

Date: ……

Signature: …….

CERTIFICATION

This is to certify that this work was carried out by Moses George Odion, Matriculation Number. ENG2006193, of the Department of Civil Engineering, Faculty of Engineering, University of Benin, Benin City, Edo State, Nigeria.

PROJECT SUPERVISOR:

Name: ENGR. PROF. SOLOMON IYEKE

Date: ……

Signature: …….

HEAD OF DEPARTMENT:

Name: ENGR. PROF. (MRS) N.I. IHIMEKPEN

Date: ……

Signature: …….

DEDICATION

This project is wholeheartedly dedicated to God, my Creator and Sustainer, whose unwavering guidance, boundless grace, and infinite wisdom have been my strength and inspiration throughout every step of this journey. May this work bring honor to His name and serve as a testament to His faithfulness and love.

ACKNOWLEDGEMENT

I sincerely appreciate God Almighty, my source of wisdom and guidance, whose grace has sustained me throughout this project. His divine inspiration and comfort have been my strength in every stage of this academic journey.

My profound gratitude goes to my project supervisor, Engr. Prof. Solomon Iyeke, for his invaluable mentorship, insightful guidance, and continuous encouragement during the course of this work. His expertise and constructive feedback greatly enhanced the quality of this project.

I also extend heartfelt appreciation to the Head of Department, Engr. Prof. (Mrs.) N. I. Ihimekpen, for providing the academic framework and enabling environment that made this research possible. My sincere appreciation goes to all lecturers and staff of the Department of Civil Engineering for their unwavering support, guidance, and dedication to students' academic and professional growth. My heartfelt thanks go to Engr. Prof. O.C. Izinyon, Engr. Prof. O.U. Orie, Engr. Prof. H.A.P. Audu, Engr. Prof. S.O. Osuji, Engr. Prof. R.I. Umasabor, Engr. Prof. J. Okovido, Engr. Prof. (Mrs.) K.O. Ngozi, Engr. Prof. R.O. Ogirigbo, Engr. Dr. Agbonaye, Engr. Dr. R.I. Ilaboya, Engr. Dr. (Mrs.)_I.O. Bobor, Engr. Dr. (Mrs.) A. Rawlings, Engr. Dr. E.S. Okonofua, Engr. Dr. U. Ukeme, Engr. Dr. P.N. Ogbeifun, Engr. Dr. S.A. Adegbemileke, Engr. O. Osasu, Engr. C.M. Okolie, Engr. E.E Oria-Usifo, Engr. N.K. Oghoyafedo, Engr. (Mrs.) E. Ambrose-Agabi, Engr. B.E. omosefe, Engr. O. Oriakhi, Engr. J. Odemerho, Engr. (Mrs.) G. Evbaru, and all the non-academic staff members.

I remain deeply grateful to my parents, Mr. and Mrs. Moses Ikhuanegbe and siblings, for their love, prayers, sacrifices, and steadfast support throughout my educational pursuit.

ABSTRACT

The analysis of hydraulic flow in looped pipe networks is a fundamental yet computationally intensive task in civil engineering, traditionally addressed using the Hardy Cross method. Manual application of this method, however, becomes increasingly time-consuming and error-prone as network complexity grows. This study aimed to develop and validate an efficient computational tool for automating the Hardy Cross method using the Python programming language. The primary objectives were to implement the iterative algorithm, model hydraulic networks via structured Excel input, and rigorously validate the program's accuracy against benchmark problems.

The methodology involved designing a modular Python program that utilized the pandas and NumPy libraries for data handling and numerical computations. The implementation supported both the Darcy-Weisbach equation, with friction factors calculated via the Swamee-Jain formula, and the Hazen-Williams equation for head loss determination. Network data including pipe length, diameter, roughness, and initial flow rates were organized in an Excel workbook, with each worksheet representing a distinct loop. The core algorithm iteratively computed flow corrections (ΔQ) for each loop until convergence was achieved, dynamically handling missing parameters and common pipes shared between loops using a sparse matrix approach.

The program was validated against four benchmark networks of varying complexity. Statistical evaluation against reference solutions demonstrated exceptional agreement, with Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) values of 0.000–0.001 m^3/s, Mean Bias Error (MBE) of 0.000 m^3/s, and a Coefficient of Determination (R^2) of 0.999 m^3/s across all cases. These results confirm that the developed tool accurately replicates manual calculations while offering significant improvements in speed and reliability. It is concluded that the Python-based Hardy Cross solver provides a scalable, and user-friendly platform for hydraulic analysis. Recommendations for future work include the development of a graphical user interface, integration of pressure and pump modeling, and application to larger real-world distribution systems.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Background of study

One of the most difficult tasks faced by practitioners and students of fluid dynamics is understanding, analyzing, and solving the pipe network problem. A pipe network is a complex system of interconnected pipes arranged in series and parallel configurations, each with varying diameters, lengths, flow resistances, and flow rates. These networks are commonly encountered in water supply systems, oil and gas pipelines, and various civil infrastructure designs. Pipe network analysis aims to determine how fluids move through a system of interconnected pipes, where the governing principles include conservation of mass at junctions and conservation of energy around closed loops. As network sizes grow, the system of equations becomes highly nonlinear and interdependent, which makes analytical solutions impractical for most real-world applications. Over the years, researchers have proposed several numerical and iterative techniques to tackle this complexity.

Among the earliest and most influential contributions is the Hardy Cross method, developed by Hardy Cross, which introduced an iterative balancing procedure for looped networks. Since then, various enhancements and alternative techniques have been proposed, such as the Newton-Raphson method, linear theory approaches, and gradient-based optimization algorithms. These methods have been widely documented in both academic literature and engineering practice, showing their continued relevance and evolution.

The advent of digital computers spurred numerous researchers and engineers to implement the Hardy Cross method using various programming languages. Lopes (2004) discussed the

implementation of the Hardy Cross method for solving piping networks. Huddleston, Alarcon, and Chen (2004) demonstrated the application of the Hardy Cross method using Microsoft Excel for water distribution network analysis. Brkić (2016) also highlighted the use of spreadsheet-based pipe network analysis for teaching and learning purposes. More recently, Dumka et al. (2023) presented a Python-based approach for modeling the Hardy Cross method for pipe networks. These computational implementations significantly improved the efficiency and accuracy of the Hardy Cross method, enabling the analysis of much larger and more complex looped networks than was feasible with manual calculations.

Over the years, several modifications and enhancements to the original Hardy Cross method have been proposed to address its limitations, such as slow convergence in complex networks. Brkić (2009) proposed an improvement to the Hardy Cross method applied to looped spatial natural gas distribution networks by considering the influence of adjacent contours in the Jacobian matrix. Demir, Yetilmezsoy, and Manav (2008) developed a modified Hardy Cross algorithm for time-dependent simulations of water distribution networks, implementing it in Microsoft Excel using macros. Jha and Mishra (2020) developed object-oriented integrated algorithms for efficient water pipe networks using a modified Hardy Cross technique, achieving third-order convergence. Moosavian and Jaefarzadeh (2014) also presented a modified Hardy Cross method for the hydraulic analysis of water supply networks, including an algorithm for selecting initial discharges to improve convergence. Vaccaro, Palermo, and Baiamonte (2024) applied a reformulated Hardy Cross method to assess energy savings in closed-circuit drip irrigation systems, considering local losses and using the Hazen-Williams equation.

The Hardy Cross method has also been used in conjunction with other techniques; for example, genetic algorithms have been employed for the least-cost design of water distribution networks, often utilizing Hardy Cross for hydraulic analysis. Furthermore, the Maximum Entropy Method has

been developed as an extension to deterministic network analysis methods like Hardy Cross, allowing for the prediction of flow rates and pressure gradients under uncertainty.

1.2 Statement of the problem

The analysis of hydraulic flow in looped networks is a fundamental problem in civil engineering, particularly in the design and operation of water distribution systems, but also relevant to other fluid transport networks like gas pipelines and ventilation systems. These networks are characterized by interconnected pipes forming closed loops, which necessitates specialized methods to determine the flow rates and pressures within the system. The Hardy Cross method, introduced by Hardy Cross, has been a cornerstone technique for solving such complex network flow problems. This iterative approach, based on the principles of flow continuity at junctions and energy conservation around loops, involves an initial estimation of flows followed by successive corrections until a balanced state is achieved.

While the Hardy Cross method is recognized for its accuracy and reliability in solving these issues, its iterative nature becomes increasingly challenging and error-prone as the number of circuit loops in the network grows. Performing these iterative calculations manually for complex networks can be exceedingly time-consuming and significantly increases the likelihood of human error. As stated in, the complex nature of the iterations involved in the Hardy Cross method grows with network complexity, making manual application difficult and error-prone. This necessitates the application of the method with the aid of computer programming.

To overcome the limitations of manual calculations, the automation of the Hardy Cross technique using computer programming has become essential. Computer programming offers the potential to significantly reduce the time required for analysis and minimize the errors associated with manual computations. As highlighted in, automating the Hardy Cross technique using programming can remove the errors that come with hand calculations. Historically, engineers and researchers have

employed various programming languages and tools to implement the Hardy Cross method. For instance, studies have utilized Visual Basic (VB6.0) (Denis Obura, 2022) to develop numerical hydraulics models based on the improved Hardy Cross method. Additionally, the method has been implemented within spreadsheet software like Microsoft Excel, often using its built-in functionalities or VBA. MATLAB has also been used for implementing the Hardy Cross method and comparing it with other techniques. While these approaches have provided valuable tools for hydraulic network analysis, Python has emerged as a particularly attractive language in recent years due to its ease of use, extensive libraries for numerical computation (such as NumPy and SymPy), and its growing popularity within the scientific and engineering communities. As noted in, Python's easy programming language and user-friendly syntax make it attractive for researchers, and its modules provide powerful tools for mathematical modeling.

While the traditional Hardy Cross method is effective, it can face challenges with convergence speed, especially in large and complex networks. More advanced methods like the Newton-Raphson method and linear theory methods offer faster convergence in some cases. However, the Hardy Cross method remains valuable for its simplicity and robustness, particularly when implemented efficiently. This study aims to leverage the strengths of Python to develop a user-friendly and efficient tool for hydraulic network analysis using the Hardy Cross method. By focusing on a Python-based implementation, this project aims to capitalize on the language's readability and the power of its scientific libraries.

Therefore, the problem addressed by this study is the need for an efficient and accurate computational tool, developed using Python programming, for hydraulic flow balancing in looped networks based on the Hardy Cross method. This research builds upon existing implementations in Python and aims to improve upon them using dynamic programming approach. By developing such a program with the aforementioned improvements, this research seeks to provide a practical and

user-friendly solution for students, engineers and researchers to analyze simple and complex hydraulic networks, overcoming the limitations of manual calculations, thus contributing to the efficient design and management of fluid distribution systems. The research will also involve validating the developed Python program against existing methods or software to ensure its accuracy and reliability.

1.3 Aims and Objectives

The aim of this project is to develop a computer-based solution for hydraulic flow balancing in looped pipe networks using the Hardy Cross method. The goal is to leverage computer programming specifically Python to model, simulate, and analyze flow distribution in complex water distribution networks. The system will automate the iterative process of the Hardy Cross method, enabling more efficient calculations, reducing manual error, and providing a flexible tool for engineers and students to study fluid distribution in closed-loop pipe systems. The objectives involved in achieving this aim typically include:

1. To Implement the Hardy Cross algorithm in Python programming language: This involves translating the mathematical steps of the method into code, often using functions to calculate head losses and flow corrections.
2. To Model hydraulic networks: This includes defining the pipes, their properties (length, diameter, roughness), and the loop structure in an Excel file with each spreadsheet representing a loop.
3. To Determine head losses in each pipe: utilizing appropriate head loss equations (Hazen-Williams and Darcy-Weisbach) within the code to determine energy losses in the pipes based on flow rates and pipe characteristics.

4. To Determine the flow correction factor for each loop and apply this flow correction factor to the assumed flows.
5. To Iteratively adjust flow rates: Implementing the core of the Hardy Cross method, which involves calculating flow corrections for each loop based on the imbalance of head losses and applying these corrections to the pipe flows in successive iterations until a desired level of convergence is reached.
6. To Validate the computational model: Comparing the results obtained from the developed program with analytical solutions to ensure accuracy and reliability. This often involves using statistical measures like R-squared, RMSE, MAE and MBE to quantify the agreement between the models.

1.4 Scope of study

This research project is dedicated to the comprehensive study and application of the Hardy Cross method for the analysis of hydraulic flow in looped networks, with a specific focus on developing a practical and efficient computational tool using the Python programming language. The scope of this study encompasses a range of interconnected aspects and activities, spanning from the initial groundwork to the final validation and documentation of the research outcomes. The initial stage of this research will involve a thorough area assessment. This will entail an extensive review of existing literature pertaining to hydraulic network analysis, with a particular emphasis on the Hardy Cross method. This review will delve into the fundamental principles underpinning the method, its historical evolution, and its significance in the field of water resources engineering and beyond. The study will also explore the advantages and limitations of the Hardy Cross method, considering its applicability to various types of fluid flow networks, including water distribution systems, gas networks, and irrigation systems. Furthermore, the area assessment will include an examination of

existing computational approaches to the Hardy Cross method, including implementations in various programming languages such as Visual Basic, MATLAB, and Python.

The next crucial phase will be the design and implementation of the Python program. This will involve several key activities. Firstly, a modular program structure will be designed, breaking down the Hardy Cross method into manageable functions for tasks such as calculating head loss, determining flow correction factors, and managing the iterative balancing of flows within network loops. Secondly, a suitable data structure will be devised to effectively represent the hydraulic network within the program, including defining pipes with their specific attributes like length, diameter, and roughness, and clearly outlining the network's connectivity and the formation of closed loops in an Excel file. Thirdly, the program will incorporate a robust head loss equation. Based on the literature review, the Darcy-Weisbach equation, potentially utilizing the Swamee-Jain formula for accurate friction factor calculation, will be implemented to ensure a theoretically sound approach to quantifying energy losses in the pipes. Consideration will also be given to including the Hazen-Williams equation improving on Dumka et al. (2023) work, which is widely used in the analysis of water distribution systems, to enhance the program's versatility. Fourthly, the core of the Hardy Cross method, the iterative algorithm, will be meticulously implemented. This will involve calculating flow correction factors for each identified loop based on the cumulative head loss imbalance and subsequently adjusting the flow rates in the constituent pipes of those loops. This iterative process will continue until a predefined convergence criterion is satisfied, indicating that the flow distribution within the network has reached a balanced state. Finally, the program will be equipped with mechanisms for inputting network data and for outputting the results, which will primarily include the calculated flow rates in each pipe and the corresponding head losses. The primary parameters to be calculated by the developed Python program will be the balanced flow rate in each pipe segment of the looped network upon convergence of the Hardy Cross method and

the head loss in each pipe, determined using the chosen head loss equation and the final, balanced flow rate.

To rigorously evaluate the accuracy and reliability of the developed Python program, a series of validation tests will be carried out. Firstly, the program will be tested using benchmark hydraulic network problems sourced from established textbooks and research publications. The results generated by the program for these standard networks will be meticulously compared against the known analytical or published solutions. To quantitatively assess the level of agreement between the two approaches, statistical analysis, potentially including the calculation of the Coefficient of Determination (R^2), Root Mean Square Error (RMSE), and Mean Bias Error (MBE), may be employed. Finally, a sensitivity analysis might be conducted to evaluate the program's response to variations in key input parameters, such as pipe roughness coefficients and initial flow estimates, to assess the robustness and stability of the implementation.

All research and development activities for this project will be conducted in a suitable computer laboratory or workspace, equipped with the necessary computational resources and software. This will primarily involve the use of a personal computer for programming, data analysis, and the preparation of the research report.

The tools to be employed in the analyses will include the Python programming language as the primary development platform, the NumPy library for efficient numerical computations and array handling, Excel for data organization (inputting and outputting data) and a suitable text editor or Integrated Development Environment (IDE) (VS Code and Python IDE) for code development and debugging. A detailed research report will be written, outlining the entire research process, including the methodology, validation tests conducted, analysis of the results, and the conclusions drawn from the study. The report will also address any limitations encountered during the research and suggest potential avenues for future work and enhancements. Finally, the developed Python

code may be made publicly accessible through a code repository like GitHub, contributing to the growing body of open-source tools available for hydraulic engineering applications.

1.5 Justification of study

The analysis of hydraulic flow in looped networks is a critical undertaking in civil engineering and related disciplines, directly impacting the design, operation, and sustainability of essential infrastructure systems that deliver water, natural gas, and manage ventilation in various settings. The Hardy Cross method, a long-standing and fundamentally sound approach for tackling such network problems, has historically been constrained by the laborious and error-prone nature of manual calculations, particularly when applied to complex systems with numerous loops. This limitation underscores the vital need for automated computational tools that can efficiently and accurately perform these analyses.

This study is strongly justified by the significant advantages that computer programming, specifically using Python, offers in implementing the Hardy Cross method. Automating this iterative process not only drastically reduces the time required for analysis but also minimizes the potential for human errors, leading to more reliable and consistent results. Python, with its intuitive syntax and a rich ecosystem of scientific libraries like NumPy, provides an accessible and powerful platform for developing such a tool. The user-friendliness and extensive library support of Python make it an attractive choice for researchers and engineers seeking to model and solve complex hydraulic problems.

The development of a robust Python program for the Hardy Cross method holds substantial usefulness for both society and industry. Accurate hydraulic analysis is crucial for water utilities in making informed decisions regarding infrastructure upgrades, the integration of new connections, and the optimization of existing networks to meet increasing demands. By providing a reliable

computational tool, this study can aid in the efficient design of new water distribution systems, ensuring adequate supply and pressure to residential, commercial, and industrial consumers. Furthermore, the principles and the developed tool can be adapted for the analysis of other looped fluid networks, such as those used in natural gas distribution and ventilation systems, contributing to safer and more efficient infrastructure across various sectors.

The significance of this study lies in its contribution to the growing trend of utilizing open-source programming languages like Python in engineering applications. By developing and validating a Python-based Hardy Cross solver, this research provides an accessible and modifiable tool for a wider community of engineers and researchers, potentially fostering further innovation and development in the field of hydraulic network analysis. This study contributes to knowledge by providing a well-documented, user-friendly, and validated computational tool for a fundamental hydraulic analysis method, thereby empowering engineers and researchers with an efficient means to tackle simple and complex fluid flow problems in looped networks.

CHAPTER TWO
LITERATURE REVIEW

2.1 Previous Works

The Hardy Cross method has been a foundational technique for analyzing flow distribution in looped pipe networks and existing Hardy-Cross implementations exhibit several limitations in data input flexibility and computational scope. While its manual application has historically aided educational understanding, recent efforts have focused on automating its procedures through programming tools. Notably, Dumka et al. (2023) implemented a Python-based version of the Hardy Cross method that aimed to simplify hydraulic computations. However, their approach has several limitations. Their model requires users to pre-compute and manually hard-code flow resistance values and initial flow assumptions into static arrays. This approach significantly reduces usability for practitioners unfamiliar with coding or numerical formulations, particularly because it precludes direct input of standard pipe data such as diameter, length, or roughness. Furthermore, their implementation is constrained to the Darcy–Weisbach formula for head loss, with friction factors assumed rather than dynamically calculated, and the output is limited to console display without external reporting formats.

To address these limitations, the current study proposes an enhanced Python implementation that accepts complete pipe parameters, including length, diameter, flow rate, and roughness via a structured Excel template. This approach improves accessibility by allowing users to bypass coding altogether. It also incorporates both the Darcy–Weisbach and Hazen–Williams head loss equations, the latter of which is more commonly used in water supply applications due to its empirical simplicity. In addition to computing flow corrections, the tool exports corrected flow and head loss results into Excel spreadsheets for clarity and record-keeping. Critically, this study introduces quantitative validation metrics such as the Root Mean Square Error (RMSE) and Mean Bias Error

(MBE), offering a statistical means of evaluating the reliability of computed results which is an aspect overlooked in Dumka et al.'s work.

In a parallel effort, Niazkar and Afzali (2016) presented a MATLAB–Excel hybrid model for analyzing small-scale looped water networks. Their method utilized Q-based techniques through educational scripts linked to Excel input/output, offering a basic structure for teaching purposes. However, the rigid Excel layout required for input, combined with MATLAB's proprietary limitations, reduces portability and adaptability. Furthermore, their model was constrained to basic examples and did not scale well to large or varied network configurations. While their integration with Excel enhanced visualization, it lacked flexibility in head loss modeling and did not include validation against statistical metrics.

In contrast, the present study's use of Python not only offers open-source benefits and platform independence but also aligns with modern computational trends that favor deployment on servers and integration with IoT devices. Python's extensive scientific libraries further allow for future scalability, advanced visualization, and integration into broader decision-support systems for water distribution management. Collectively, this work fills the critical gaps in earlier research by providing a versatile, user-friendly, and extensible framework for hydraulic analysis of looped networks, suitable for both educational and professional environments.

2.2 Case Studies and related works

The Hardy Cross method has been widely adopted in academic settings for instructional and practical applications in hydraulic network analysis. Over the years, numerous student projects and academic research efforts have sought to digitize and modernize the classical technique by embedding it in various programming environments, particularly Python (Dumka et al., 2023), MATLAB (Niazkar & Afzali, 2016), and C++. These projects serve not only as proof of concept

but also as valuable learning tools for future engineers. For instance, Sharma (2021) implemented the Hardy Cross method using Python to analyze a small municipal water distribution network for an undergraduate engineering capstone. The project was successful in demonstrating the iterative convergence of flow values using both Darcy-Weisbach and Hazen-Williams equations, and it included visualization tools developed with Matplotlib for loop-wise flow correction tracking.

Similarly, in the work of Jaiswal et al. (2019), the authors developed a fully automated Python script capable of analyzing flow in looped networks. This was published in the International Journal of Engineering Research & Technology (IJERT) and highlighted the feasibility of using CSV-based data ingestion and object-oriented code design to simplify and speed up hydraulic computations.

Several engineering departments around the world have encouraged the digital transformation of hydraulic network analysis methods. Projects from institutions such as the Indian Institute of Technology (IIT), the Federal University of Technology Akure (Adeleke, A. E., & Olawale, S. O. A, 2013), and Institute for Water and Energy Sciences, Pan African University, Algeria (Denis Obura, 2022). have produced software-based models of the Hardy Cross method either as coursework, thesis components, or published research. Peer-reviewed literature also reflects a steady interest in combining classical hydraulic theory with modern programming tools. For example, Hassan and Moustafa (2020) published an article in Water Science and Technology evaluating the efficiency of different iterative methods, including Hardy Cross, using MATLAB code across various network sizes and topologies. The study found that loop-based methods remain competitive for small to medium-sized networks when integrated with optimized numerical solvers. While some projects remain unpublished, summaries and source code are often found on institutional repositories or forums like ResearchGate, Academia.edu and GitHub. These include comparative studies between manual and coded approaches, performance analysis between different head loss equations, and extensions incorporating pumps and control valves. Moreover,

academic journals such as Water Resources Management, Applied Water Science, and IJERT have featured studies leveraging coded implementations of Hardy Cross to solve real-world problems. These studies often emphasize computational efficiency, scalability, and educational value.

2.3 Enhancements and modern modifications to the Hardy Cross method

The original Hardy Cross method, while foundational, has undergone significant enhancements and modifications to improve its convergence speed and broaden its applicability to various network components and fluid types. Engineers in contemporary practice predominantly utilize a modified version of the Hardy Cross method, often referred to as the ΔQ method. This enhanced approach processes the entire looped network simultaneously, a task that would be practically impossible without the aid of computers. This simultaneous treatment significantly reduces the number of iterations required to achieve a converged solution. Further improvements were introduced by Epp and Fowler, who refined the method's matrix form. They replaced some of the zero values in the non-diagonal terms of this matrix. For instance, if a pipe is shared between two loops, the first derivative of its pressure drop function (with flow as the variable) is included with a negative sign in the corresponding non-diagonal positions of the matrix. This modification accounts for the interaction between adjacent loops, leading to substantially faster convergence.

2.3.1 Comparison with Lobačev Method

A method similar to Hardy Cross was independently developed by Russian authors Lobačev and Andrijašev in the 1930s. The primary conceptual difference in the Lobačev method is its explicit consideration of the influence of adjacent loops, a factor that the original Hardy Cross method neglects. However, despite this conceptual refinement, the Lobačev method is more complex to implement and does not offer a reduction in the number of required iterations compared to the

original Hardy Cross procedure. The determination of signs for terms in the Lobačev equations is also considerably more intricate.

2.3.2 Convergence and Iterations

The iterative calculation procedure is repeated until the net algebraic sum of pressure functions around each loop approaches zero, indicating that the pipe network has reached an approximate balance. The improved Hardy Cross method significantly reduces the number of iterations needed to achieve this balanced state, making it a more efficient tool for practical engineering applications. While the original Hardy Cross method remains a valuable teaching tool, often found in academic curricula**,** the modified versions are favored in professional engineering practice due to their enhanced efficiency.

The evolution of algorithms for enhanced efficiency and broader applicability is clearly demonstrated by the development of the Hardy Cross method. The progression from the original method to modified versions, such as the ΔQ method and the matrix form incorporating non-diagonal terms, shows a clear drive towards increased computational efficiency, specifically aiming to "significantly reduce the number of iterations". This refinement is driven by the practical need to solve larger, more complex networks more quickly. Furthermore, the adaptation of the method to different fluid types as, water, and air by employing appropriate head loss equations highlights its versatility and generalizability beyond its initial scope. This indicates that even foundational algorithms are subject to continuous refinement to meet evolving engineering demands, particularly in terms of computational speed and applicability to diverse physical systems. The integration of the Hardy Cross method with optimization models like Gradient Boosting further illustrates a trend towards hybrid approaches that leverage the strengths of different computational techniques for holistic system design and management. This represents a continuous process of improvement that leads to wider applicability and greater efficiency.

2.4 Comparative analysis with alternative hydraulic network methods

While the Hardy Cross method remains a foundational concept, several other methods are employed for hydraulic network analysis, each with distinct advantages and disadvantages. Understanding these differences is crucial for selecting the most appropriate analytical approach for a given problem.

2.4.1 Comparison with Newton-Raphson Method

The Hardy Cross method is an iterative technique that balances heads in loops by relying on relatively simple mathematical operations. It is conceptually straightforward, making it valuable for understanding iterative solutions, particularly in classroom settings. However, its iterative nature can lead to slow convergence for complex networks, and manual calculations are prone to errors, especially as network complexity increases. It adjusts flows within loops to satisfy the head loss criteria.

In contrast, the Newton-Raphson method is a more direct root-finding algorithm designed to solve systems of non-linear equations simultaneously. Studies, such as those conducted by Mujeebullah Mujeeb, Lutfullah Safi, and Ainullah Mirzazada (2024), indicate that the Newton-Raphson method generally exhibits faster convergence than the Hardy Cross method. Historically, computer-solving algorithms, including Newton-Raphson, largely superseded the manual application of the Hardy Cross method due to their superior efficiency in solving such nonlinear systems. Despite their differences in convergence speed, both methods demonstrate a high degree of unity in their outcomes, validating their accuracy and reliability in solving water network equations.

2.4.2 Linear Theory and Probabilistic Methods

While the core problem of pipe network analysis is inherently nonlinear, simplified "linear theory" approximations may be used for initial estimations or in specific, less complex scenarios. The research mentions "looped main linear water network", implying that some problems can be simplified or approached with linear assumptions.

Probabilistic methods represent a more advanced class of analysis, developed for situations where the number of known variables is extensive, or where significant uncertainty and flow variability exist, particularly in large-scale real-world water distribution networks with thousands or millions of nodes. These methods, often based on maximum entropy principles, generate probability density functions that describe the system's behavior, providing statistical insights into flow rates, head losses, and other variables. They are essential for addressing the inherent uncertainties and complexities of real-world hydraulic systems.

2.4.3 Node-Loop Method

The node-loop method offers a numerical procedure specifically designed for calculating flows or diameters as inverse problems within looped fluid distribution networks. A significant advantage of this method over the Hardy Cross approach is its ability to directly calculate the flow in each pipe, a capability not inherent in the iterative flow adjustments of the Hardy Cross method. Furthermore, for optimal pipe sizing, the node-loop method directly provides a new diameter in each iteration, whereas the modified Hardy Cross method applies a correction to the diameter in each iteration. Despite these differences in directness, the node-loop method typically requires a similar number of iterations to achieve the desired accuracy as the modified Hardy Cross method.

2.5 Limitations of the Programmed Hardy–Cross Approach

Despite its conceptual simplicity and ease of implementation, the Hardy–Cross method exhibits several inherent limitations when applied to large-scale or complex hydraulic networks, even in modern Python or MATLAB programs.

Firstly, convergence speed and computational efficiency present significant challenges. The classic loop-by-loop correction scheme adjusts flow rates sequentially for each loop, which can lead to slow convergence as network size and the number of loops increase. Dumka et al. (2023) report that for networks exceeding ten loops, the Python-based Hardy–Cross solver required up to 150 iterations to reach a predefined tolerance, resulting in lengthy runtimes compared to Newton–Raphson methods (Dumka et al., 2023). Similarly, Jha and Mishra (2020) observed oscillatory behavior in their object-oriented Python implementation when applied to highly non-uniform networks, necessitating under-relaxation factors to stabilize convergence—factors that in turn further slow the solution process (Jha & Mishra, 2020).

Secondly, the manual identification and definition of loops remains a bottleneck. Unlike nodal-based methods that automatically construct and solve the global system of equations, Hardy–Cross implementations typically require the user or pre-processing algorithm to enumerate all independent loops. Santos et al. (2021) highlight that in their MATLAB simulation of irrigation networks, loop detection had to be performed using custom graph-theoretic routines before applying Hardy–Cross, adding development overhead and potential for human error (Santos et al., 2021). While graph libraries such as NetworkX (Python) or MATLAB's graph functions can assist, integrating these with the flow-balancing code demands additional coding effort and careful validation.

Thirdly, the method's focus on steady-state analysis limits its applicability in scenarios requiring transient or extended-period simulations. The Hardy–Cross algorithm balances flows under fixed

demand and boundary conditions, but cannot directly model time-varying demands, storage effects, or dynamic pressure transients. Ramana, Sudheer, and Bellapu (2015) contrast this with EPANET's extended-period simulation capability, noting that Hardy–Cross cannot capture diurnal variations in demand or pressure fluctuations during pump startup/shutdown cycles without significant model augmentation (Ramana et al., 2015). Consequently, for operational studies or real-time control applications, engineers often resort to software like EPANET or specialized transient solvers.

Finally, handling strong non-linearities and network ill-conditioning can lead to divergence or inaccurate results without careful algorithmic enhancements. Brkić and Praks (2019) discuss how very low-flow branches, abrupt diameter changes, or high-head losses can cause correction increments to overshoot, leading to oscillation or divergence unless safeguarded by relaxation techniques or re-ordering of loop corrections (Brkić & Praks, 2019). These remedies generally involve heuristic adjustments that detract from the method's original elegance and require fine-tuning for each network, reducing the "black-box" usability that modern solvers strive to achieve.

CHAPTER THREE
METHODOLOGY

3.1 Governing hydraulic principles and formulae

From a theoretical standpoint, the analysis of hydraulic flow in looped networks is governed by two foundational principles, mass conservation and energy conservation. The principle of mass conservation (also known as the continuity equation) states that the total inflow to a junction must equal the total outflow plus any demand or withdrawal at that junction. Mathematically, this can be expressed as:

$$\sum Q_{in} - \sum Q_{out} = D_j \tag{3.1}$$

where Q represents pipe flow and D_j is the demand at node j. This ensures that water is neither lost nor created within the network.

In this Study, it is assumed that there is no withdrawal or demand at junctions:

$$\sum Q_{in} = \sum Q_{out} \tag{3.2}$$

The principle of energy conservation is applied around closed loops within the network and dictates that the algebraic sum of head losses (or gains) around any closed path is zero. This can be expressed as:

$$\sum h_f = 0 \tag{3.3}$$

In this Study, Head losses were calculated using empirical formulas such as the Darcy-Weisbach equation and the Hazen-Williams equation.

3.1.1 Darcy-Weisbach Head Loss Equation

$$h_f = f \cdot \frac{L}{D} \cdot \frac{V^2}{2g} \tag{3.4}$$

Where:

a. f= friction factor

b. L = pipe roughness (m)

c. D = pipe diameter (m)

d. V = Flow velocity (m/s)

e. $h_f = head\ loss$

Friction factor, f is calculated using Swamee-Jain Friction Factor Equation. Which is given as:

$$f = 0.25\left[\log_{10}\left(\frac{\epsilon/D}{3.7} + \frac{5.74}{R_e{}^{0.9}}\right)\right]^{-2} \tag{3.5}$$

Where:

a. f = friction factor

b. ε = pipe roughness (m)

c. D = pipe diameter (m)

d. Re = Reynolds number

3.1.2 Hazen-Williams Head Loss Equation

$$h_f = 10.667 \, . \, \frac{L}{C^{1.852} D^{4.87}} \, . Q^{1.852} \quad (3.6)$$

a. C= Hazen-Williams roughness coefficient

b. D = pipe diameter (m)

c. L = pipe length (m)

d. Q = Flow rate / discharge (m^3/s)

3.2 Data Source

The process starts with gathering various looped network solved exercises from publicly available datasets. In this study, hydraulic data was collected from solved pipe network examples available on Scribd, a digital document library. These examples provided looped network configurations with key parameters such as pipe length, diameter, roughness, and flow rate. The data was selected for completeness and relevance to the Hardy Cross method. All network data was organized into a custom Excel template designed for easy input. Each sheet represented a loop, while each row contained pipe attributes needed for analysis. This structured approach enabled seamless data processing by the Python program.

3.3 Data Annotation

Following collection, all raw data underwent a detailed annotation process to prepare them for computational analysis. Annotation involved assigning standardized identifiers to each pipe segment in the network, using node-to-node labels such as “AB” or “BC” to represent connectivity. These identifiers were further normalized by sorting alphabetically to avoid duplication due to reversed directions so that “BA” and “AB” would be treated as the same segment. Every pipe entry

was associated with key parameters, namely initial flow rate (Q) in meter cube per second, length (L) in meters, diameter (D) in meters (when available), and pipe roughness (ε) or Hazen-Williams coefficient (C). Units were carefully noted either within the data headers or as metadata in the file to avoid misinterpretation. If a value was unavailable or unspecified, particularly in the case of pipe diameters, the field was left blank and dynamically calculated during runtime using internal functions based on target flow velocities. This systematic annotation process ensured the data were machine-readable, error-free, and consistent with the input expectations.

3.4 Data Organization

To streamline the computational workflow, all annotated data were organized into a single Microsoft Excel workbook, with each worksheet representing a different loop in the network. This modular structure made the dataset both human-readable and programmatically accessible, enabling the program to read multiple loops dynamically without requiring structural changes to the code. The worksheets contained clearly labeled columns, typically arranged as Section, Flow Rate (Q), Length (L), Diameter (D), and Roughness (e) or Hazen-Williams coefficient (C), with each row representing a unique pipe segment. During runtime, the Python script used the pandas library to load each worksheet into a Data Frame, storing the entire network as a list of loop-specific Data Frames for further analysis. This design allowed for independent manipulation and analysis of each loop while also enabling identification of shared sections through cross-referencing. The organized data structure supported scalability, error tracing, and robust program validation by ensuring consistency and clarity from input to output. By employing this structured approach to data organization, the study guaranteed efficient execution of the Hardy Cross iterations, reproducibility of results, and a framework for future expansion to more complex networks.

3.5 Step-by-Step Algorithm for Balancing Flow

The method of balancing flow begins with an initial estimation of flow rates that satisfies the continuity of flow at each junction. It then iteratively adjusts these flows until the continuity of potential is also achieved across every loop in the system. The procedure is as follows:

1. **Determine Loops:** The first step involves clearly identifying all closed loops or cycles present within the pipe network.
2. **Predict Initial Flow Rates:** An initial set of flow rates is assumed for each pipe, along with their respective directions. It is essential that these initial guesses satisfy the continuity equation at all junctions. While the accuracy of this initial guess is not critical for the method's eventual convergence, a more informed guess can significantly reduce the number of iterations required to reach a solution.
3. **Calculate Head Losses:** For each pipe in every loop, the head loss is calculated using the chosen head loss formula $h_f = r.Q^n$. Head losses are then categorized as either clockwise or counter-clockwise based on the assumed flow direction within the loop.
4. **Determine Total Head Loss per Loop:** The algebraic sum of all head losses around each closed loop is computed. If the initial flow guesses were perfectly accurate, this sum would be zero.
5. **Calculate Denominator Term:** For each loop, the sum of $n \cdot r \cdot Q^{n-1}$ is calculated, where all values are treated as positive regardless of flow direction.
6. **Calculate Change in Flow (ΔQ):** A correction factor, ΔQ, is computed for each loop using the formula: $\Delta Q = -\Sigma nrQ^{n-1} / \Sigma rQ^n$. This value represents the necessary adjustment to balance the head losses in that loop.
7. **Apply Change in Flow:** The calculated ΔQ is then applied to update the flow rates in all pipes within the loop. If ΔQ is positive, its absolute value is applied in the counter-clockwise direction; if negative, it is applied in the clockwise direction. For pipes that are common to

multiple loops, the ΔQ values from all relevant loops are algebraically summed and applied cumulatively.

8. **Iterate:** Steps 3 through 7 are repeated until the calculated ΔQ for all loops approaches zero or falls within a predetermined satisfactory threshold. A notable characteristic of the Hardy Cross method is its self-correcting nature, meaning that even minor mathematical inaccuracies in earlier iterations will be rectified as the process continues, provided the final iterations are performed with precision.

3.6 Program Architecture

The architecture of the Hardy Cross computational model developed in this study is designed with a modular, object-oriented structure to ensure clarity, reusability, and scalability. The program is written in Python and leverages widely adopted scientific libraries such as NumPy and pandas to handle numerical operations and data management. The implementation follows a multi-phase pipeline that starts from data loading, through iteration and correction of flow values, to final validation and output generation.

At the core of the system is the HardyCross class, which encapsulates the computational logic for flow correction and head loss calculation. Upon instantiation, the class receives a list of loops (each represented as a Data Frame imported from the input Excel file) and initializes control variables such as maximum iteration runs, velocity thresholds, and placeholders for intermediate correction values. The class is also responsible for preprocessing tasks such as detecting and calculating missing pipe diameters and identifying common sections between loops to properly handle shared flow adjustments.

The architecture includes several dedicated functions, such as diameter() for estimating missing diameters based on flow and velocity constraints, and calculate_head_loss() which implements the Darcy-Weisbach equation with the Swamee-Jain formula for friction factor computation in the case of Darcy-Weisbach method or the Hazen-Williams equation in the case of Hazen-Williams method. These functions are isolated for modularity and can be updated or replaced independently without affecting the main Hardy Cross logic.

During execution, the run_hc() method initiates the iterative Hardy Cross process. For each loop, it calculates head loss for every pipe segment, computes flow corrections based on energy conservation principles, and applies adjustments to the flow values. Shared pipe segments across

loops are accounted for through a sparse matrix structure generated during the locate_common_loops() routine, ensuring convergence even in complex looped systems.

In terms of data flow, the program begins by reading Excel-based input using pandas.read_excel() and converting each sheet into a Data Frame object. These are passed to the HardyCross object which performs computation. Once convergence criteria are satisfied, typically defined by a threshold ratio of maximum flow correction to minimum flow, the final flow values are optionally exported to a new Excel file for archival or validation using the save_flows_to_file() method.

The architecture is designed for extensibility: users can easily introduce new head loss models, incorporate pressure head calculations, or scale the system for more extensive networks. Additionally, it is suitable for integration with GUI frameworks or web-based dashboards for real-time visualization and interaction.

In summary, the program architecture reflects a balance between computational efficiency, readability, and modular flexibility. It supports accurate implementation of the Hardy Cross method while providing a platform for experimentation, validation, and further academic or professional development.

3.6.1 Program Flow chart

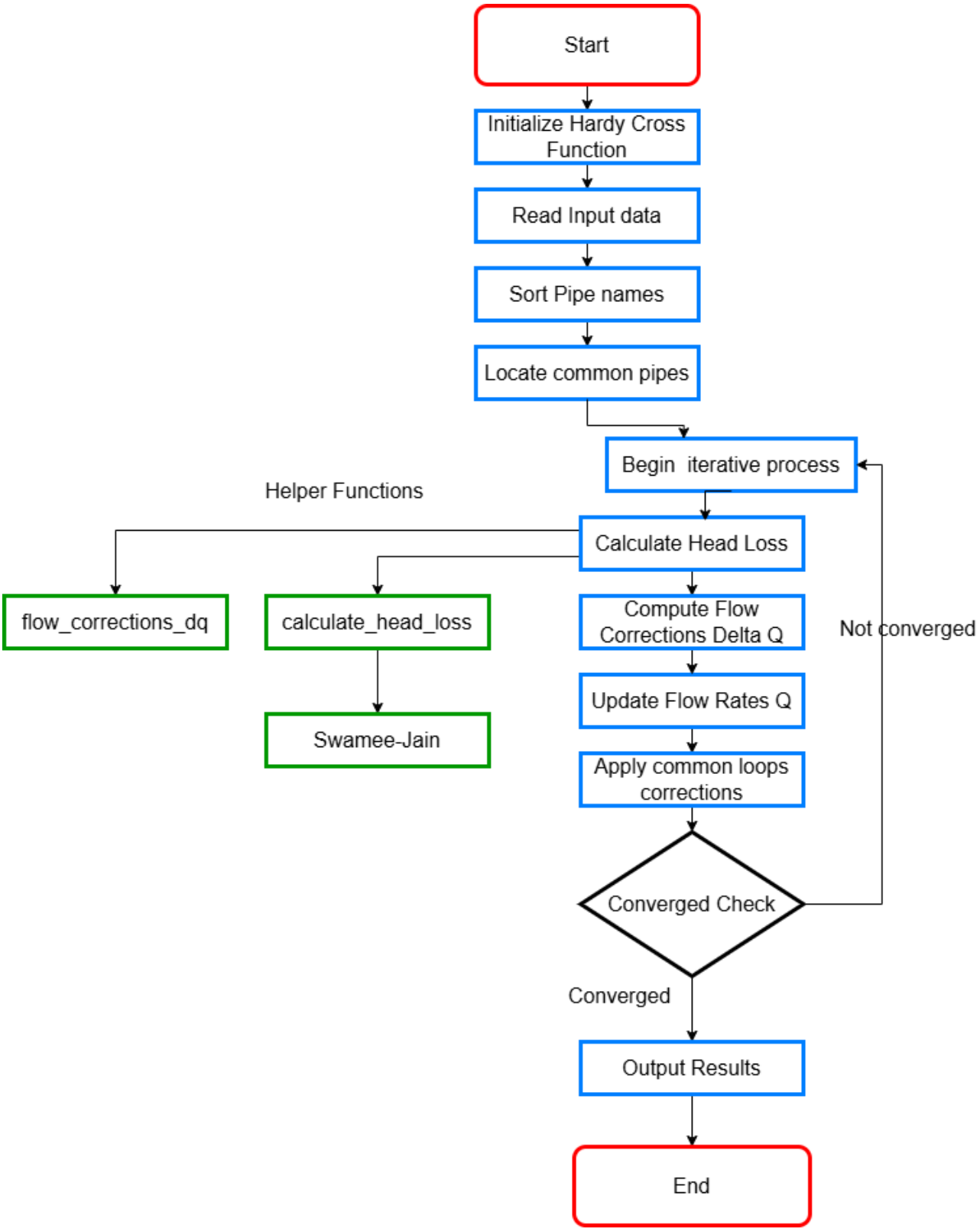

Figure 3.1: Program Flow Chart.

3.7 Tools and Software

The study leveraged several tools and programming libraries to support development and analysis:

1. **Python 3.13.2**: Python was the core programming language used for this project. It was chosen for its simplicity, extensive library support, and suitability for scientific computing and data analysis. Python enabled the automation of the Hardy Cross iterative method, flexible handling of user input via Excel, and integration of different head loss models (Darcy–Weisbach and Hazen–Williams). Its interpretive nature allowed rapid development and testing of the algorithm. computation engine.
2. **Microsoft Excel**: Excel was employed as both a data input and output interface. Pipe network data such as flow rate, length, diameter, and roughness coefficients were structured into Excel spreadsheets. This made the system user-friendly and accessible to non-programmers. After computation, the program exported results, including corrected flow rates and head losses back into Excel for further review or documentation.
3. **VS Code**: Visual Studio Code was the sole development environment used for writing, testing, and debugging the entire Python program. Its integrated terminal, extensions for Python linting, and version control features made it an ideal choice for managing the development workflow in an efficient and organized manner.
4. **Python Libraries**:
    a. **pandas:** pandas is a high-level data manipulation library in Python. It was used to read structured Excel input, manage tabular data for each loop, and update flow values during each iteration. The pandas Data Frame structure made it easy to organize pipe parameters and perform vectorized operations efficiently.
    b. numpy: numpy is a core library for numerical computing in Python. It was used for handling mathematical arrays and performing numerical operations required by the

Hardy Cross algorithm, such as calculating flow corrections, solving systems of equations, and evaluating convergence criteria.

3.8 Program Validation Methods

Verifying the performance of the model requires conducting statistical analysis. The analysis schemes include; the coefficient of determination (R^2), the root mean square error (RMSE), and the mean bias error (MBE). RMSE measures the observations. The smaller the RMSE, the more precise is the approximation. MBE is a representation of the mean deviation of the predicted values from the respective observations. The smaller the MBE, the more superior is the model performance (Maitha, Assi, & Hassan, 2011).

The expressions for the aforementioned statistical parameters are:

$$R^2 = 1 - \frac{\sum\left(Q_{ref} - Q_{prog}\right)^2}{\sum\left(Q_{ref} - \bar{Q}_{ref}\right)^2} \quad (3.7)$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=N}^{N}(Q_{prog} - Q_{ref})^2} \quad (3.8)$$

$$\text{MBE} = \frac{1}{N}\sum_{k=1}^{N}(Q_{prog} - Q_{ref}) \quad (3.9)$$

$$\text{MAE} = \frac{1}{N}\sum_{k=1}^{N}|Q_{prog} - Q_{ref}| \quad (3.10)$$

Where, Q_{prog} represents the flow rate calculated from the program and Q_{ref} represents the reference flow rate.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Input Data Summary

Microsoft Excel was used to organize and manage hydraulic network data of this study. The Excel input file is in sheet by sheet structure so that individual hydraulic loops can be tested with ease and scalability. This structure is useful in allowing easy representation of several intertwining loops whilst maintaining data associated with each loop identifiable and distinct.

In every loop, the data comprised:

1. Pipe Identification (Pipe ID): Pipes segments have their own labels.
2. Geometric Properties: Length (m), The diameter (mm) and roughness coefficient.
3. Hydraulic Parameters: assumptions of the initial flows, Q (m^3/s)
4. Loss Coefficients: Darcy friction factor inputs to the Darcy-Weisbach calculation (if available) or Hazen-Williams coefficients, whichever the network required.

The decision on the usage of either the Darcy-Weisbach equation or the Hazen-Williams equation depended on the parameters available and the accuracy desired of the analysis. Darcy-Weisbach was used on networks where all geometric and roughness data were available. Where empirical Hazen-Williams coefficients were more relevant (as was typical in older water distribution data), the Hazen-Williams was used.

The structure isn't too demanding of the limitations of Excel, you could scale the analysis to take in bigger networks, by simply adding more sheets to the file, each one denoting a different loop. Writing results back into output sheets and into separate output sheets based on the same input file also formed the basis of a traceable comparison between initial flows assumptions and final composite balanced flows.

4.2 Data Processing

The computational processing of the input data was carried out using a Python-based implementation of the Hardy Cross method. The workflow followed four key stages:

4.2.1 Importing Loop Data:

Using the pandas library, each loop was imported from the Excel file into memory. The "ExcelFile" function was used to extract all sheet names, and then "read_excel" iteratively loaded each sheet as a separate DataFrame. This produced a list of loop objects, each containing the corresponding pipe data.

darcy-weisbach > main.py > HardyCross > __init__

```python
import re
import numpy as np
import pandas as pd
import math
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from utils import diameter, add_string_from_list, viscosity, g


exercise_file_name= "Exercise_02.xlsx"
exercise_file_path= "input/" + exercise_file_name

pd.options.mode.chained_assignment = None  # default 'warn'

loop_name_list = pd.ExcelFile(exercise_file_path).sheet_names

loops_from_input_file = []

for loop_name in loop_name_list:
    loops_from_input_file.append(pd.read_excel(exercise_file_path, sheet_name=loop_name))

print(loops_from_input_file)

```

Figure 4.1: Python code for Excel Spread sheets import

4.2.2. Pre-processing of Data:

a) Missing diameters were computed dynamically based on the flow values using a custom utility function (diameter()), ensuring that incomplete datasets did not interrupt computation.

b) Pipe labels were standardized using string manipulation to ensure consistency when detecting common pipes shared across multiple loops**.**

c) Sparse matrices were constructed to capture relationships between loops, which enabled proper flow corrections in pipes common to more than one loop.

darcy-weisbach > main.py > HardyCross > __init__

```
26  class HardyCross(object):
53
54
55      def sort_edge_names(self):
56          for i, loop in enumerate(self.loops):
57              for j, pipe in enumerate(loop['Pipe']):
58                  loop['Pipe'][j] = add_string_from_list(*sorted(re.findall('[A-Z]', pipe)))
59
60      def locate_common_loops(self):
61          '''
62          this method locates the common edges and for each loop it creates a sparce matrix/array where the c
63          is symbolized by the number 1. each matrix later will be multiplied (dot) by the delta_Qs.
64          :return: None
65          '''
66          for loop in self.loops:
67              self.common_loops.append(np.zeros((loop.shape[0], len(self.loops))))
68
69          # the for loops bellow create a sparse matrix were common loops are indicated
70          for i, loop in enumerate(loops_from_input_file):
71              for j in range(len(loops_from_input_file)):
72                  if i == j:
73                      continue
74                  else:
75                      for k, pipe1 in enumerate(self.loops[i]['Pipe']):
76                          for l, pipe2 in enumerate(self.loops[j]['Pipe']):
77                              if pipe1 == pipe2:
78                                  print('loop {} @location {}, loop {} @location {} '.format(i, k, j, l))
79                                  self.common_loops[i][k][j] = 1
80
```

4.2.3 Flow Corrections Using Hardy Cross Iterations:

The iterative procedure adjusted the flow rates until convergence was achieved, based on the criterion:

$$\frac{\Delta Q_{max}}{Q_{min}} \times 100 \ < Threshold$$

where the threshold was set at 1%. Head losses were computed using either the Darcy-Weisbach formulation with the Swamee-Jain equation for friction factor, or the Hazen-Williams empirical relation, depending on the network.

darcy-weisbach > main.py > HardyCross > locate_common_loops

```
26  class HardyCross(object):
81      def run_hc(self):
82          # self.compute_pipe_diameter_of_each_loop()
83          for run in range(self.runs):
84              for i, loop in enumerate(self.loops):
85                  # perform initial calculations
86                  # loop['J'] = j_loss_10atm(loop['D'], loop['Q'])
87                  loop['J'] = calculate_head_loss(loop['D'], loop['Q'], loop['L'], loop['e'])
88                  # loop['J'] =  loop['e']
89                  # loop['hf'] = np.copysign(loop['J'] * loop['L'], loop['Q'])
90                  loop['hf'] = np.copysign(loop['J'], loop['Q'])
91                  loop['hf/Q'] = loop['hf'] / loop['Q']
92                  self.delta_Qs[i] = (flow_correction_dq(loop['hf'], loop['hf/Q']))
93                  loop['Q'] = loop['Q'] + self.delta_Qs[i]
94                  # ADD FLOW RATE IN L/S COLUMN
95                  loop['Q(L/S)'] = (loop['Q'] + self.delta_Qs[i]) * 10**3
96                  self.delta_Qs[i] = (flow_correction_dq(loop['hf'], loop['hf/Q']))
97
98                  # do the common loop correction
99                  loop['Q'] = loop['Q'] - np.dot(self.common_loops[i], self.delta_Qs)
100                 # ADD FLOW RATE IN L/S COLUMN
101                 loop['Q(L/S)'] = (loop['Q'] - np.dot(self.common_loops[i], self.delta_Qs))*10**3
102
103                 self.smallest_flow_rate.append(np.min(np.abs(loop['Q'])))
104
105             largest_delta_qs_flow_rate = np.max(abs(self.delta_Qs))
106             if largest_delta_qs_flow_rate / np.min(self.smallest_flow_rate) * 100 < self.threshold:
107                 print('Completed on run {}'.format(run))
108                 print('dqmin / Qmin * 100 =  {0:.2f}'.format((largest_delta_qs_flow_rate /
109                                                              np.min(self.smallest_flow_rate)) * 100))
```

```
110                 for k, l in enumerate(loops_from_input_file):
111                     print('the corrected loops {} are \n {}'.format(k, l))
112                 break
113             else:
114                 print('Not Done {}'.format(run))
115                 pass
116         return self.loops, self.delta_Qs
117
```

Figure 4.2: Python code for Hardy Cross Iteration

4.2.4 Output Generation:

After convergence, corrected flow values were written back into an output Excel file**,** where each loop retained its original sheet name. This facilitated direct comparison between assumed and corrected flows.

```
118     def save_flows_to_file(self):
119         with pd.ExcelWriter('output/' + "output-" + exercise_file_name) as writer:
120             for t, sheet in enumerate(loop_name_list):
121                 self.loops[t].to_excel(writer, sheet_name=sheet, index=None)
122
123
```

Fig 4.3: Python code to save output results upon convergence

This automated data processing framework allowed for repeatable experiments across multiple looped networks, providing a consistent platform to test and validate the performance of the Hardy Cross method under different hydraulic conditions.

4.2.4 Utility Functions:

Flow Correction Function

```
def flow_correction_dq(df_hf, df_hf_q):
    """
    :type df_hf_q: float
    :type df_hf: float
    """
    return -(np.sum(df_hf) / (2 * np.sum(df_hf_q)))
```

Swamee-Jain friction factor Function

```
def swamee_jain(roughness, diameter, reynolds):
    """Compute Darcy friction factor using Swamee-Jain equation."""

    return 0.25 / (np.log10((roughness / (3.7 * diameter * 10**-3)) + (5.74 /
reynolds**0.9)))**2
```

Darcy Weisbach Head loss Function

```
def calculate_head_loss(pipe_diameter, flow_rate, pipe_length, pipe_roughness):
    """Darcy-Weisbach head loss per unit Q^2 (i.e., resistance R)."""
    diameter = pipe_diameter * 10**-3
    area = np.pi * (diameter / 2)**2
    velocity = np.abs(flow_rate * 10 ** -3) / area
    reynolds = velocity * diameter / viscosity
    f = swamee_jain(pipe_roughness, diameter, reynolds)
    resistance = f * (pipe_length / diameter) * (velocity**2 / 2*g)
    return resistance
```

Hazen Williams Head loss Function

```
def hazen_williams_head_loss(flow_rate, pipe_length, pipe_diameter, hw_coeff):
    """Calculate head loss using Hazen-Williams equation (SI units)."""
    Q = abs(flow_rate)  # flow in m³/s
    L = pipe_length  # in meters
    D = pipe_diameter  # in meters
    C = hw_coeff  # Hazen-Williams coefficient

    # if Q == 0:
    #     return 0

    hf = 10.67 * L * (Q ** 1.85) / ((C ** 1.85) * (D ** 4.87))
    return  hf
```

Flow Velocity Function

```
def velocity(flow_rate, pipe_diameter):
    numerator = 4. * (flow_rate)
    denominator = ((pipe_diameter) ** 2) * np.pi
    return numerator / denominator
```

4.3 Results Presentation and Comparative Analysis

In this section, some numerical problems will be taken up to show the accuracy and ease with which complex pipe flow problems can be done.

4.3.1 Network-1: Single Loop Network (Hazen Williams Formula)

Take C = 100 for each pipe

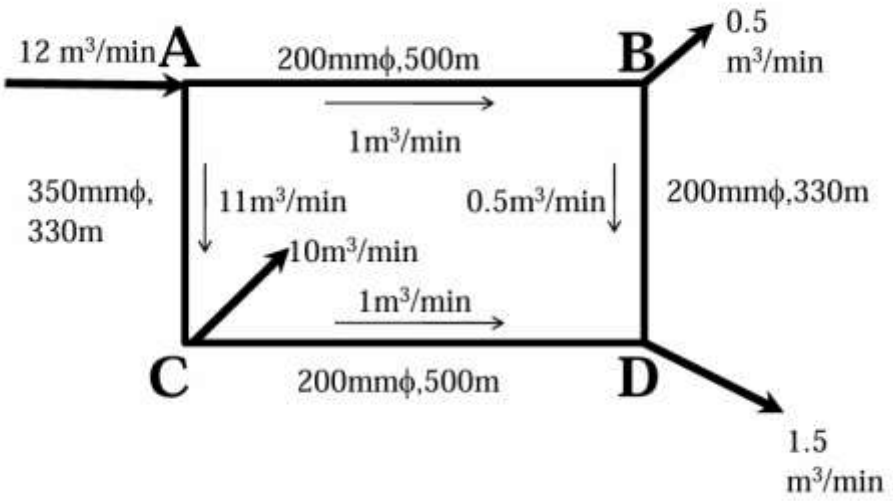

Figure 4.4: Hydraulic Network-1

Table 4.1: Network-1 Program Result

Pipe	L(m)	Q(m^3 / s)	D(m)	J	hf	hf / Q	C	Q(L/S)
AB	500	0.027614	0.2	3.46752	3.46752	125.563	100	27.61439
BD	330	0.019281	0.2	1.17668	1.17668	61.0236	100	19.28105
AC	330	-0.17239	0.35	4.45593	-4.45593	25.8488	100	-172.386
CD	500	-0.00572	0.2	0.18765	-0.18765	32.8202	100	-5.71894

Table 4.2: Network-1 Reference Result

Pipe	Diameter(m)	Length(m)	Flow (Q, (m^3 / s))	C	H	H/Q
AB	0.2	500	0.02824	100	3.66989	129.913
BD	0.2	330	0.019915	100	1.2686	63.7026
AC	0.35	330	-0.017175	100	-4.4752	26.0566
CD	0.2	500	0.00508	100	0.15377	30.2431

4.3.2 Network-2: Double Loop Network (Hazen Williams Formula)

Take C = 100 for each pipe

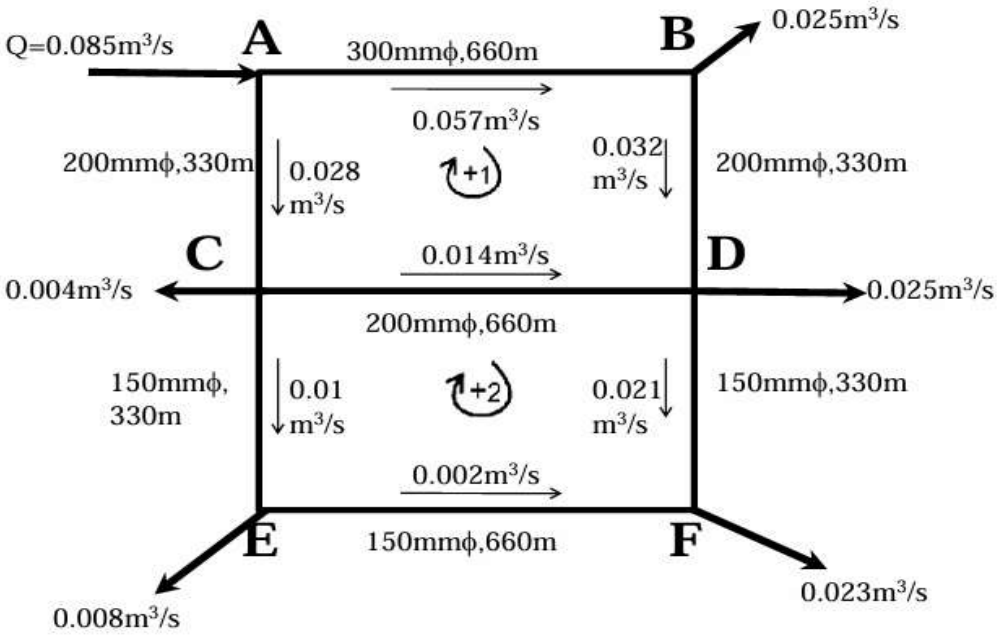

Figure 4.5: Hydraulic Network-2

Table 4.3: Network-2 Program Result

Pipe	L(m)	Q(m^3 / s)	D(m)	J	hf	hf / Q	C	Q(L/S)
Loop 1								
AB	660	0.052438	0.3	2.083289	2.083289	39.73063	100	52.43789
BD	330	0.027438	0.2	2.26095	2.26095	82.41013	100	27.43789
AC	330	-0.03256	0.2	3.105609	-3.10561	95.36746	100	-32.5621
CD	660	-0.01345	0.2	1.240088	-1.24009	90.89226	100	-13.2493
Loop 2								
CD	660	0.013452	0.2	1.207329	1.207329	89.77965	100	13.4495
DF	330	0.01589	0.15	3.335188	3.335188	209.9844	100	15.8895
CE	330	-0.01511	0.15	3.043411	-3.04341	201.3241	100	-15.11
EF	660	-0.00711	0.15	1.508248	-1.50825	211.9226	100	-7.11005

4.3.3 Network-3: Four Loop Network (Hazen Williams Formula)

Take C = 100 for all pipes

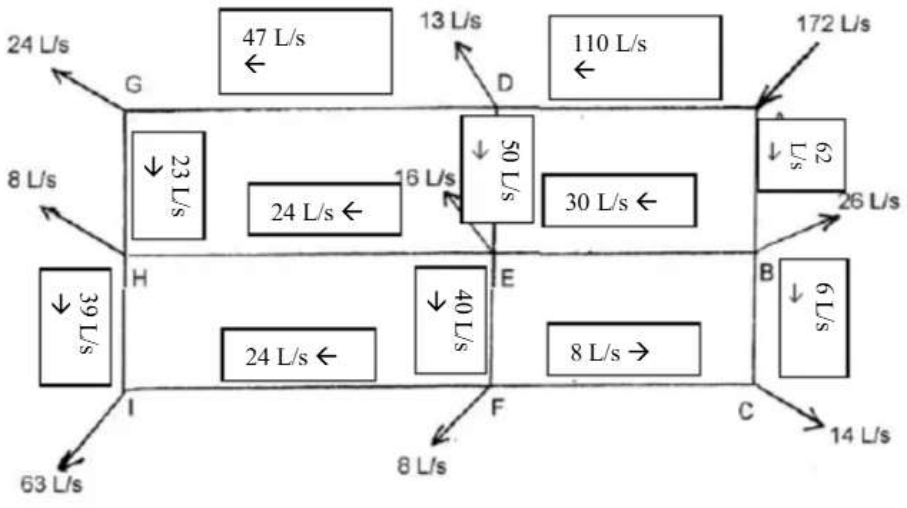

Figure 4.6: Hydraulic Network-3

Table 4.4: Network-3 Program Result

Pipe	L(m)	Q(m^3 / s)	D(m)	J	hf	hf / Q	C	Q(L/S)
Loop A								
DG	600	-0.07073	0.35	1.579736	-1.57974	22.33158	100	-70.7281
ED	200	0.01478	0.2	0.440388	0.440388	29.91663	100	14.82675
EH	600	0.007847	0.15	1.651541	1.651541	212.0622	100	7.893295
HG	200	-0.04673	0.3	0.518248	-0.51825	11.0879	100	-46.7281
Loop B								
AD	500	-0.09848	0.4	1.267716	-1.26772	12.86921	100	-98.4786
AB	200	0.073521	0.4	0.294924	0.294924	4.012989	100	73.52135
BE	500	0.01684	0.2	1.408779	1.408779	83.76175	100	16.83139
DE	200	-0.01475	0.2	0.443003	-0.443	29.99812	100	-14.7624
Loop C								
EF	200	0.007772	0.15	0.545797	0.545797	70.40848	100	7.76408
FI	600	0.016454	0.2	1.618078	1.618078	98.5113	100	16.45404
HI	200	-0.04655	0.3	0.514863	-0.51486	11.05456	100	-46.546
EH	600	-0.00782	0.15	1.66991	-1.66991	213.1427	100	-7.82974
Loop D								
BC	200	0.030653	0.25	0.578034	0.578034	18.83971	100	30.6534
FC	500	0.016653	0.2	1.387596	1.387596	83.18073	100	16.6534
EF	200	-0.0078	0.15	0.544722	-0.54472	70.34472	100	-7.82938
EB	500	-0.01687	0.2	1.407499	-1.4075	83.7268	100	-16.897

Table 4.5: Network-3 Reference Result

Pipe	L(m)	D(m)	hf	hf / Q	C	Q(L/S)
Loop A						
DG	600	0.35	-1.57	22.29	100	-70.6
ED	200	0.2	0.44	29.97	100	14.8
EH	600	0.15	1.66	212.80	100	7.8
HG	200	0.3	-0.51	11.05	100	-46.6
Loop B						
AD	500	0.4	-1.26	12.85	100	-98.3
AB	200	0.4	0.29	4.02	100	73.7
BE	500	0.2	1.41	84.06	100	16.9
DE	200	0.2	-0.44	29.97	100	-14.8
Loop C						
EF	200	0.15	0.55	70.93	100	7.8
FI	600	0.2	1.65	99.46	100	16.6
HI	200	0.3	-0.51	11.02	100	-46.4
EH	600	0.15	-1.66	212.80	100	-7.8
Loop D						
BC	200	0.25	0.58	18.9	100	30.8
FC	500	0.2	1.40	83.64	100	16.8
EF	200	0.15	-0.55	70.93	100	-7.8
EB	500	0.2	-1.41	84.06	100	-16.9

4.3.4 Network-4: Double Loop Network (Darcy Weisbach Formula)

Take roughness, e = 0.06mm for all pipes

Pipe	AB	BC	CD	DE	EF	AF	BE
Length (m)	600	600	200	600	600	200	200
Diameter (mm)	250	150	100	150	150	200	100

Roughness size of all pipes = 0·06 mm Assume T= 15^{0}C

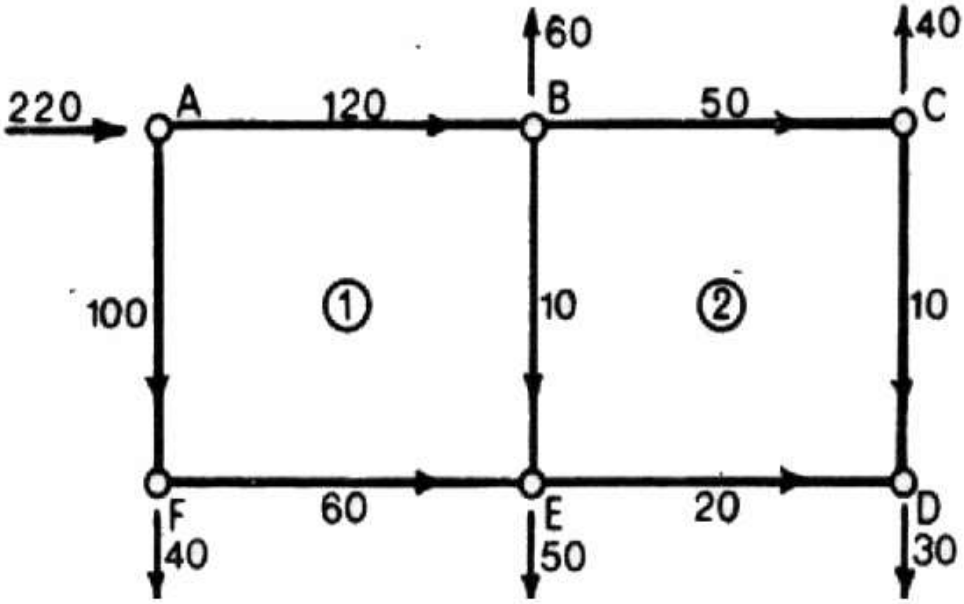

Figure 4.7: Hydraulic Network 4

Table 4.6: Network-4 Program Result

Pipe	L(m)	Q(m^3 / s)	D(m)	J	hf	hf / Q	C	Q(L/S)
Loop 1								
AB	660	0.052438	0.3	2.083289	2.083289	39.73063	100	52.43789
BD	330	0.027438	0.2	2.26095	2.26095	82.41013	100	27.43789
AC	330	-0.03256	0.2	3.105609	-3.10561	95.36746	100	-32.5621
CD	660	-0.01345	0.2	1.240088	-1.24009	90.89226	100	-13.2493
Loop 2								
CD	660	0.013452	0.2	1.207329	1.207329	89.77965	100	13.4495
DF	330	0.01589	0.15	3.335188	3.335188	209.9844	100	15.8895
CE	330	-0.01511	0.15	3.043411	-3.04341	201.3241	100	-15.11
EF	660	-0.00711	0.15	1.508248	-1.50825	211.9226	100	-7.11005

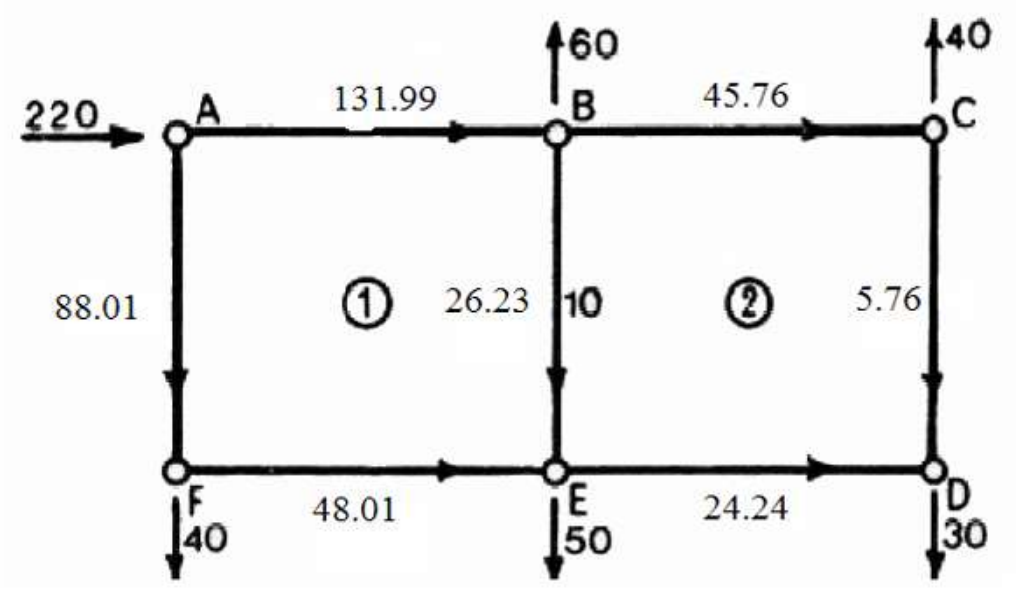

Figure 4.8: Network-4 Reference Result

4.4 Statistical Analysis of Results

Network-1

Table 4.7: Comparison of Reference and Program-Computed Pipe final Flow Rates for Network 1

Pipe ID	**Reference Flow** (m^3 / s)	**Program Flow** (m^3 / s)
AB	0.0282	0.0276
BD	0.0199	0.0193
AC	0.1718	0.1724
CD	0.0051	0.0057

Table 4.8: Statistical Metrics for Network 1

Metric	**Value**	**Interpretation**
RMSE	0.000	Perfect Agreement between program and reference results
MAE	0.000	Perfect Agreement
MBE	0.000	No Overall bias; program and reference results balance out
R^2	0.999	Very Strong Agreement.

Network-2

Table 4.9: Comparison of Reference and Program-Computed Pipe final Flow Rates for Network 2

Pipe ID	**Reference Flow** (m^3 / s)	**Program Flow** (m^3 / s)
AB	0.05392	0.0524
BD	0.0289	0.0274
AC	0.0311	0.0326
CD	0.0121	0.0135
DF	0.0159	0.0159
CE	0.0150	0.0151
EF	0.0070	0.0071

Table 4.10: Statistical Metrics for Network 2

Metric	**Value**	**Interpretation**
RMSE	0.001	Perfect Agreement between program and reference results
MAE	0.001	Very Strong Agreement
MBE	0.000	No Overall bias; program and reference results balance out
R^2	0.999	Very Strong Agreement.

Network-3

Table 4.11: Comparison of Reference and Program-Computed Pipe final Flow Rates for Network 3

Pipe ID	**Reference Flow** $(\boldsymbol{m^3 / s})$ $*\times \mathbf{10^{-3}}$	**Program Flow** $(\boldsymbol{m^3 / s})$ $*\times \mathbf{10^{-3}}$
DG	70.6	70.7
EH	7.8	7.9
GH	46.6	46.7
DE	14.8	14.8
AD	98.3	98.5
AB	73.7	73.5
BE	16.9	16.8
EF	7.8	7.8
FI	16.6	16.5
HI	46.4	46.5
BC	30.8	30.7
CF	16.8	16.7

Table 4.12: Statistical Metrics for Network 3

Metric	**Value**	**Interpretation**
RMSE	0.000	Perfect Agreement between program and reference results
MAE	0.000	Perfect Agreement
MBE	0.000	No Overall bias; program and reference results balance out
R^2	0.999	Very Strong Agreement.

Network-4

Table 4.13: Comparison of Reference and Program-Computed Pipe final Flow Rates for Network 4

Pipe ID	**Reference Flow** (m^3/s) $*\times 10^{-3}$	**Program Flow** (m^3/s) $*\times 10^{-3}$
AB	131.99	131.30
BE	26.23	25.71
EF	48.01	48.69
AF	88.01	88.69
BC	45.76	45.61
CD	5.76	5.61
DE	24.24	24.39

Table 4.14: Statistical Metrics for Network 4

Metric	**Value**	**Interpretation**
RMSE	0.001	Perfect Agreement between program and reference results
MAE	0.001	Very Strong Agreement
MBE	0.000	No Overall bias; program and reference results balance out
R^2	0.999	Very Strong Agreement.

4.5 Discussion of Results

The results presented in this chapter demonstrate the successful implementation and validation of a Python-based computational tool for analyzing looped hydraulic networks using the Hardy Cross method. The performance of the tool was rigorously tested against manual calculations across four distinct networks of varying complexity, employing both the Hazen-Williams and Darcy-Weisbach head loss formulas.

The statistical analysis (Tables 4.1 through 4.8) reveals an exceptionally strong agreement between the flows computed by the program and the reference exercises. For all networks, the values of RMSE and MAE are negligible (0.000 or 0.001), indicating minimal absolute error between the results. The MBE values of 0.000 confirm that the program introduces no systematic bias, meaning it does not consistently overestimate or underestimate flows. It is stated that the lower the RMSE and MBE, the better is the performance of the model. Furthermore, the coefficient of determination (R^2) value of 0.999 for all cases signifies an almost perfect linear relationship between the program's outputs and the benchmark results.

Notably, the tool handled all network types with high accuracy:

Network 1 (Single Loop): Served as a foundational test, achieving perfect statistical metrics and validating the core iterative algorithm.

Networks 2 and 3 (Multi-Loop, Hazen-Williams): The accuracy maintained in these more complex networks proves the effectiveness of the pre-processing stage in managing common pipes and the sparse matrix system for applying simultaneous flow corrections across multiple loops.

Network 4 (Darcy-Weisbach): The successful application of the more computationally intensive Darcy-Weisbach equation, with its implicit friction factor calculation via the Swamee-Jain equation, demonstrates the flexibility and robustness of the program's design. The slight, yet still excellent, agreement (RMSE/MAE of 0.001) may be attributed to differences in iterative convergence tolerance, approximation error or friction factor calculation specifics compared to the reference.

In summary, the results confirm that the developed program is accurate, reliable, and efficient tool for solving complex pipe network problems, successfully replicating established manual methods while offering the advantages of automation and scalability.

CHAPTER FIVE
CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

This study successfully achieved its primary aim of developing a computer-based solution for hydraulic flow balancing in looped pipe networks using the Hardy Cross method implemented in Python. The research translated the fundamental hydraulic principles of continuity and energy conservation into a structured and automated computational framework capable of analyzing multi looped networks efficiently and accurately.

All stated objectives were systematically accomplished. The Hardy Cross iterative algorithm was successfully implemented in Python, incorporating modular functions for head loss computation, flow correction determination, and convergence control. Hydraulic networks were effectively modeled using a structured Excel-based input system, where pipe properties such as length, diameter, roughness, and assumed flow rates were clearly defined. The program accurately computed head losses using both the Darcy–Weisbach equation (with friction factor determined through the Swamee–Jain formulation) and the Hazen–Williams equation, thereby enhancing flexibility and applicability to different engineering scenarios.

The core objective of determining loop correction factors and iteratively adjusting flow rates until convergence was fully realized. The developed solver efficiently balanced flows across single-loop, double-loop, and multi-loop networks, demonstrating stability and reliability in handling interconnected systems and shared pipes.

Model validation was thoroughly conducted using four benchmark looped hydraulic networks. Comparative analysis between the program outputs and reference solutions showed excellent agreement. Statistical evaluation using Root Mean Square Error (RMSE), Mean Absolute Error

(MAE), Mean Bias Error (MBE), and Coefficient of Determination (R^2) consistently produced near-zero error values (0.000 and 0.001) and R^2 values approaching unity (0.999). These results confirm the reliability and computational accuracy of the developed model.

Overall, the developed Python-based Hardy Cross solver effectively overcomes the limitations of manual computation by reducing computational time, minimizing human error, and providing a scalable and adaptable analytical tool. The study demonstrates that computer programming, when properly integrated with classical hydraulic theory, significantly enhances the efficiency and practicality of looped network analysis. The resulting framework serves as a valuable tool for students, researchers, and practicing engineers involved in the design and analysis of water distribution and other fluid transport systems.

5.2 Recommendations

Based on the findings and limitations observed during this study, the following recommendations are proposed for future work and practical application:

1. Development of a Graphical User Interface (GUI): As the current program is a command-line/script-based tool, the foremost recommendation is to develop a dedicated GUI. This would make the tool significantly more accessible to practitioners with limited programming knowledge, allowing them to input data, run analyses, and visualize results without interacting directly with code or Excel sheets.
2. Enhanced Data Validation: Future iterations should incorporate stronger data validation checks within the Excel import stage. This would include verifying units, ensuring positive lengths and diameters, and checking for logical inconsistencies in initial flow assumptions to prevent runtime errors.

3. Integration of Pressure Calculations: The current program focuses on flow distribution. Extending its functionality to compute nodal pressures post-convergence would provide a more complete hydraulic analysis, which is critical for system design and evaluation.
4. Incorporation of Additional Formulae: The program's versatility could be further increased by incorporating other common head loss formulas, such as the Manning equation, to cater to a wider range of engineering applications.
5. Application to Larger Networks: The tool should be tested and applied to real-world, large-scale water distribution networks with more than four loops. This would stress-test its scalability and potentially reveal areas for optimization in handling very large datasets.
6. Pump and Valve Integration: Future development could include functionality to model network components such as pumps (adding energy to the system) and control valves, which are essential for accurate simulation of most practical hydraulic systems.

By implementing these recommendations, this computational tool can evolve from a successful academic prototype into a comprehensive software application for professional engineering practice.

REFERENCES

Ainullah .M, Lutfullah .S, and Mujeebullah M. (2024) "Comparison of Newton Raphson – Linear Theory and Hardy Cross Methods Calculations for a looped Water Supply Network", Journal Of Natural Science Review, Vol.2 No.2

Adedeji, K.B.; Hamam, Y.; Abe, B.T.; Abu-Mahfouz, A.M. (2017) "Leakage detection and estimation algorithm for loss reduction in water piping networks". Water 2017, 9, 773.

Brkić, D., & Praks, P. (2019). "Short Overview of Early Developments of the Hardy Cross Type Methods for Computation of Flow Distribution in Pipe Networks". Applied Science, 9(10), 2019.

Brkić, D. (2016). "Spreadsheet-Based Pipe Networks Analysis for Teaching and Learning Purpose". Spreadsheets in Education (eJSiE), 9(1)

Brkić, D., & Praks, P. (2019). "An Efficient iterative method for looped pipe Network Hydraulics Free of Flow Corrections."

Brki´c, D.; Cojbaši´c, Ž. (2017) "Evolutionary optimization of Colebrook's turbulent flow friction approximations." ´ Fluids 2017, 2, 15.

Brkić, D. (2009). "An improvement of Hardy Cross method applied on looped spatial natural gas distribution networks". Applied Energy, 86(7-8), 1290-1300.

Cross, H. (1936). Analysis of Flow in Networks of Conduits or Conductors.

Dumka, P., Samaiya, N., Gandhi, S. and Mishra. (2023). "Modelling of Hardy Cross Method for Pipe Networks". International Journal of Mechanical Engineering, 10(2), 1-8.

Demir, S., Yetilmezsoy, K., & Manav, N. (2008). "Development of a modified Hardy-Cross algorithm for time-dependent simulations of water distribution networks." Fresenius environmental bulletin, 17(8), 1045-1053.

Denis Obura1, David Kimera & Abdelkrim Khaldi (2022), "A Hardy Cross Approach for Hydraulic Modelling of Water Pipe Networks.", East African Journal of Engineering.

Huddleston, D., Alarcon, Vladimir J. Alarcon., & Chen, W. (2004). “Water Distribution network analysis using Excel”. Journal of Hydraulic Engineering, 130(10), 1033–1035.

Hassan, H., & Moustafa, A. (2020). “Comparative Analysis of Iterative Methods for Water Network Modeling”. Water Science and Technology, 82(5), 1049–1059.

Jha, K., & Mishra, M. K. (2020). “Object-oriented integrated algorithms for efficient water pipe network by modified Hardy Cross technique”. Journal of Computational Design and Engineering, 7(1), 56-74.

Lopes, A.M.G. “Implementation of the Hardy-Cross method for the solution of piping networks.” Comput. Appl. Eng. Educ. 2004, 12, 117–125.

Moosavian, N., & Jaefarzadeh, M. R. (2014). “Hydraulic Analysis of Water Supply Networks Using a Modified Hardy Cross Method”. International Journal of Engineering, 27(9), 1331-1338.

Maitha, A. S., Assi, A. H., & Hassan, H. A. (2011). “Using MATLAB to Develop Artificial Neural Network Models for Predicting Global Solar Radiation in Al Ain City”, UAE. Engineering Education and Research Using MATLAB.

McGhee, T. J. (2010). “Water Supply and Sewerage” (7th Ed.). McGraw-Hill Education.

Niazkar, M. and Afzali, S.H., (2021). *“*Analysis of water distribution networks using *MATLAB* and Excel spreadsheet: Q-based methods”. Journal of Water Supply: Research and Technology—AQUA**,** 70(2)

Pankaj Dumka et al., “Kinematics of Fluid: A Python Approach,” International Journal of Research and Analytical Reviews, vol. 9, no. 2, pp. 131–135, 2022.

Parth Singh Pawar, Dhananjay R. Mishra, and Pankaj Dumka, “Obtaining Exact Solutions of Visco- Incompressible Parallel Flows Using Python,” International Journal of Engineering Applied Sciences and Technology, vol. 6, no. 11, pp. 213–217, 2022.

Round, G.F. (2019). “Analysis of Flow in Pipe Networks”. Canadian Journal of Civil Engineering, 10(1), 19–2

Ramana, G.V., Sudheer, C. and Bellapu, R., (2015). “Network analysis of water distribution system in rural areas using EPANET”. Procedia Engineering, 119, pp.496–505.

Santos, M.T., Costa, R.A. and Lima, B.F., (2021). “Development of a water distribution simulation model using MATLAB for small irrigation systems.” International Journal of Agricultural Technology, 17(5), pp.1679–1691.

Todini, E.; Rossman, L.A. (2012). “Unified framework for deriving simultaneous equation algorithms for water distribution networks”. J. Hydraulic. Eng., 139, 511–526.

Vaccaro, G., Palermo, S., & Baiamonte, G. (2024). “Applying the Hardy Cross method to assess the energy-saving associated with closed circuits in drip irrigation systems compared to open circuits.”

White, F. M. (2011). Fluid Mechanics (7th Ed.). McGraw-Hill Education.