1D MULTIGRID SOLVER FOR FINITE ELEMENT METHOD

By:

MOHAMAD AMIRUDDIN BIN AZHAR

(Matrix no. 143732)

Supervisor:

Dr. Muhammad Razi Bin Abdul Rahman

July 2022

This dissertation is submitted to

Universiti Sains Malaysia

As partial fulfilment of the requirement to graduate with honors degree in

BACHELOR OF ENGINEERING (MECHANICAL ENGINEERING)

School of Mechanical Engineering

Engineering Campus

Universiti Sains Malaysia

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed……………[illegible]……………… (MOHAMAD AMIRUDDIN BIN AZHAR)

Date…………………………………………………………………...(24/7/2022)

Statement 1:

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by giving explicit references. Bibliography/references are appended.

Signed……………[illegible]……………… (MOHAMAD AMIRUDDIN BIN AZHAR)

Date…………………………………………………………………...(24/7/2022)

Statement 2:

I hereby give consent for my thesis, if accepted, to be available for photocopying and for interlibrary loan, and for the title and summary to be made available outside organizations.

Signed……………[illegible]……………… (MOHAMAD AMIRUDDIN BIN AZHAR)

Date…………………………………………………………………...(24/7/2022)

ACKNOWLEDGE

First and foremost, I would like to express my gratitude to the Almighty God who is the one who has always guided me to work on the right path in life. Without His grace, this project could not become a reality. I received a lot of directly and indirectly help and guidance from many respected persons who deserve our greatest appreciation in completing my project and report. I would like to express my deepest gratitude to my supervisor, Dr Muhammad Razi Bin Abdul Rahman for giving me this opportunity to learn new things and providing a lot of advice and suggestions through numerous consultations to assist me to complete this project. His guidance has helped me in all the time of researching, programming and writing this thesis. I could not have imagined having a better advisor and mentor for my final year project. In addition, I would also like to expand my gratitude to our Final Year Project Coordinator, Dr Muhammad Fauzinizam Bin Razali who introduced the guideline and requirements needed to complete the project so that I can complete my final year project successfully. I believe that I may face a lot of obstacles while going through the project development without their guidance.

TABLE OF CONTENT

DECLARATION ii

ACKNOWLEDGE iii

TABLE OF CONTENT iv

LIST OF FIGURES vi

LIST OF TABLES viii

LIST OF ABBREVIATIONS ix

ABSTRAK x

ABSTRACT xi

CHAPTER 1 INTRODUCTION 1

1.1 Project Overview 1

1.2 Problem Statement 4

1.3 Objectives 4

1.4 Scope of the Project 4

CHAPTER 2 LITERATURE REVIEW 5

2.1 Overview 5

2.2 Multigrid Method 5

2.3 Finite Element Method 5

2.4 Combination of Multigrid Method with Finite Element Method 6

2.5 Conclusion of the Research 9

CHAPTER 3 RESEARCH METHODOLOGY 10

3.1 Overview 10

3.2 1-D Case Study: Cantilever Beam Deflection 10

3.3 Finding of MG Method Coding with the Implementation of FEM 11

3.4 Hardware and Software 13

3.5 Performance Test of Algorithm 13

3.6 Problem Detection 14

3.7 Coding Modification 16
3.8 Flowchart of the Algorithm.. 17
3.9 Coding Explanation.. 21
3.10 Memory Usage Performance 26
3.11 Plotting Graph.. 27
CHAPTER 4 RESULT AND DISCUSSION 28
4.1 Overview 28
4.2 MG Solutions of the Cantilever Beam Problems 28
4.3 Performance Test Result of Unmodified Source Code 30
4.4 Performance Test Result of Modified Source Code...................................... 32
4.5 Performance Comparison 35
4.6 Maximum Number of Elements Result... 36
4.7 Memory Usage Result... 37
CHAPTER 5 CONCLUSION AND FUTURE WORK 40
5.1 Conclusion... 40
5.2 Future Work 40
REFERENCES 41
APPENDICES 43

LIST OF FIGURES

Figure 1.1: Example of Multigrid Solution in different iterations [2]. 2

Figure 1.2: Example of 2D domain discretized by finite differences and finite elements [5]. 3

Figure 3.1: The main body program to record the time taken 14

Figure 3.2: The coding of four sections was tested 15

Figure 3.3: New function of the Gauss-Seidel Method 16

Figure 3.4: New coding for section 4... 17

Figure 3.5: Flowchart of the main program 18

Figure 3.6: Flowchart of the FEM function 19

Figure 3.7: Flowchart for Multigrid_TwoGrid function... 20

Figure 3.8: The profile function.. 26

Figure 3.9: The profiler window... 27

Figure 3.10: Additional function used for plotting graph 27

Figure 4.1: Deflection result for 50 elements 28

Figure 4.2: Deflection result for 100 elements 29

Figure 4.3: Deflection result for 150 elements 29

Figure 4.4: Graph result of performance test for unmodified code 31

Figure 4.5: The assumption of time taken for unmodified code................................. 32

Figure 4.6: Graph result of performance test for modified code 34

Figure 4.7: Comparison graph of unmodified and modified code.............................. 35

Figure 4.8: Graph result of performance test for every number of elements............... 36

Figure 4.9: Graph of memory allocated and memory freed.. 38

Figure 4.10: Profiler result for 100 elements 39

Figure 4.11: Profiler result for Animate function 39

A - 1: Main body program 43

A - 2: FEM function.. 44

A - 3: Multigrid_TwoGrid function part 1.. 45

A - 4: Multigrid_TwoGrid function part 2.. 46

A - 5: Gauss_Seidel Function 47

A - 6: Project_FineToCoarse function... 48

A - 7: gauss_seidel2 function.. 48

A - 8: Result Function... 49

A - 9: Animate function 50

LIST OF TABLES

Table 3.1: The main program code explanation 21

Table 3.2: The FEM function code explanation 22

Table 3.3: The Multigrid_TwoGrid function explanation 23

Table 3.4: The Gauss-Seidel function explanation 24

Table 3.5: The Project_FineToCoarse function explanation 25

Table 3.6: The new Gauss-Seidel function explanation 25

Table 4.1: Result of the performance test for unmodified code 30

Table 4.2: Result of the performance test for modified code....................................... 33

Table 4.3: Result of memory usage 37

LIST OF ABBREVIATIONS

MG	Multigrid
FEM	Finite Element Method
1D	One-Dimension
CPU	Central Processing Unit
RAM	Random Access Memory
USM	Universiti Sains Malaysia

ABSTRAK

Kod pengiraan menggunakan Kaedah Multigrid dengan gabungan Kaedah Unsur Terhingga untuk menyelesaikan masalah matematik agak jarang digunakan dalam domain awam. Biasanya, masalah matematik ini boleh diselesaikan hanya menggunakan satu daripada dua kaedah tersebut. Contoh kod sedemikian yang ditulis dalam bahasa pengaturcaraan MATLAB ditemui dalam repositori GitHub, di mana algoritma yang dilaksanakan adalah jauh dari prestasi optimum. Algoritma yang tersedia bagi penyelesai Multigrid dengan Kaedah Unsur Terhingga telah diubah suai dan diuji untuk prestasinya. Dua jenis pengiraan telah digunakan untuk menentukan prestasi penyelesai. Yang pertama ialah simulasi masa. Simulasi ini akan menentukan masa yang diambil untuk simulasi selesai bagi setiap bilangan unsur. Prestasi algoritma untuk setiap bilangan unsur boleh dikenal pasti daripada simulasi ini. Ujian masa pada algoritma menunjukkan semakin tinggi bilangan unsur, lebih tinggi masa yang diambil untuk menyelesaikan simulasi. Walau bagaimanapun, algoritma yang disediakan oleh repositori GitHub adalah kurang cekap. Oleh itu, beberapa pengubahsuaian kepada algoritma dibuat untuk meningkatkan prestasi penyelesai. Jenis kedua ialah simulasi memori. Prestasi memori telah diuji dengan menentukan nilai memori yang digunakan untuk setiap bilangan unsur. Walau bagaimanapun, data yang dihasilkan tidak mencukupi dan tidak tepat. Oleh itu, prestasi algoritma tidak dapat ditentukan menggunakan parameter ini. Seterusnya, algoritma telah diubah suai dengan menggunakan fungsi Gauss-Seidel baharu. Algoritma baharu juga telah diuji menggunakan simulasi masa. Kemudian, hasilnya dibandingkan dengan hasil simulasi masa sebelum pengubahsuaian. Ujian memberikan hasil yang lebih ketara daripada algoritma asal. Oleh itu, pengubahsuaian algoritma memberi impak positif kepada kecekapan penyelesai.

ABSTRACT

Computational code using the Multigrid Method with the combination of the Finite Element Method to solve a mathematical problem is quite rare in the public domain. Usually, the mathematical problem can be solved using only one of those two methods. A sample of such code written in MATLAB programming language was found in the GitHub repository, where the implemented algorithms are far from optimal. This available algorithm of the Multigrid solver with the Finite Element Method was modified and tested for its performance. Two kinds of computation have been used to determine the performance of the solver. The first one is time simulation. This simulation will determine the time taken for the simulation to complete for every number of elements. The performance of the algorithm for any number of elements can be identified from this simulation. The time test on the algorithm shows the high number of elements, the higher time taken to complete the simulation. However, the algorithm provided by the GitHub repository is less efficient. Therefore, some modifications to the algorithm are made to increase the performance of the solver. The second type is memory simulation. The memory performance was tested by determining the value of memory used for every number of elements. However, the data produced is insufficient and inaccurate. Therefore, the performance of the algorithm can not be determined using this parameter. Next, the algorithm was modified by using a new Gauss-Seidel function. The new algorithm also has been tested using time simulation. Then, the result was compared with the result of the time simulation before the modification. The test gives a more significant result than the original algorithm. Therefore, the modification of the algorithm gives a positive impact on solver efficiency.

CHAPTER 1

INTRODUCTION

1.1 Project Overview

In this life, many problems can be solved using mathematical solutions such as the numerical method, multigrid method, finite differential method, finite element method, etc. This mathematical solution has been used for hundreds of years back there, but it is so hard for humans to calculate this mathematical solution manually. The human mind's limitations prevent it from grasping the behaviour of its complex surroundings and inventions in a single action. The increasing use of computers in the current era will reduce the workload of humans in this world. The computer is one of the intelligent devices that help humans solve any type of problem. Discrete problems may now be handled quickly, even when the number of elements is quite huge, thanks to the development of digital computers [1]. All these mathematical methods will be implemented into programming languages to help humans compute the solution in the easiest and fastest way.

Mathematicians and engineers have taken various approaches to the discretization of continuous problems [1]. Mathematicians have developed general techniques for determining the stationarity of properly defined 'functionals' that can be applied directly to the differential equations that govern the problem, such as finite-difference approximations, various weighted residual procedures, and approximate techniques for determining the stationarity of properly defined 'functionals' [1]. Engineers, on the other hand, frequently take a more natural approach to the problem by drawing a parallel between real discrete elements and finite sections of a continuum domain [1].

There are many types of programming software that have been used to solve this mathematical problem, such as Python, MATLAB, C++, ANSYS, etc. In the engineering field, the common software used is MATLAB and ANSYS. This programming software provides many engineering solutions, such as displaying the stress and strain profile on the structural body of a car, displaying the heat profile on a steel bar, and so on.

The main focus of this paper is to conduct a research project about the multigrid method and the finite element method. The multigrid method is used to solve many numerical problems. Multigrid's main goal is to speed up the convergence of a basic iterative approach called relaxation, which decreases short-wavelength errors by performing a global correction of the fine grid solution approximation from time to time, which is performed by solving a coarse problem. Independent of the fine grid mesh size, this multigrid cycle often decreases all error components by a set amount far below one. The numerical solution of elliptic partial differential equations in two or more dimensions is one of multigrid's most common applications.

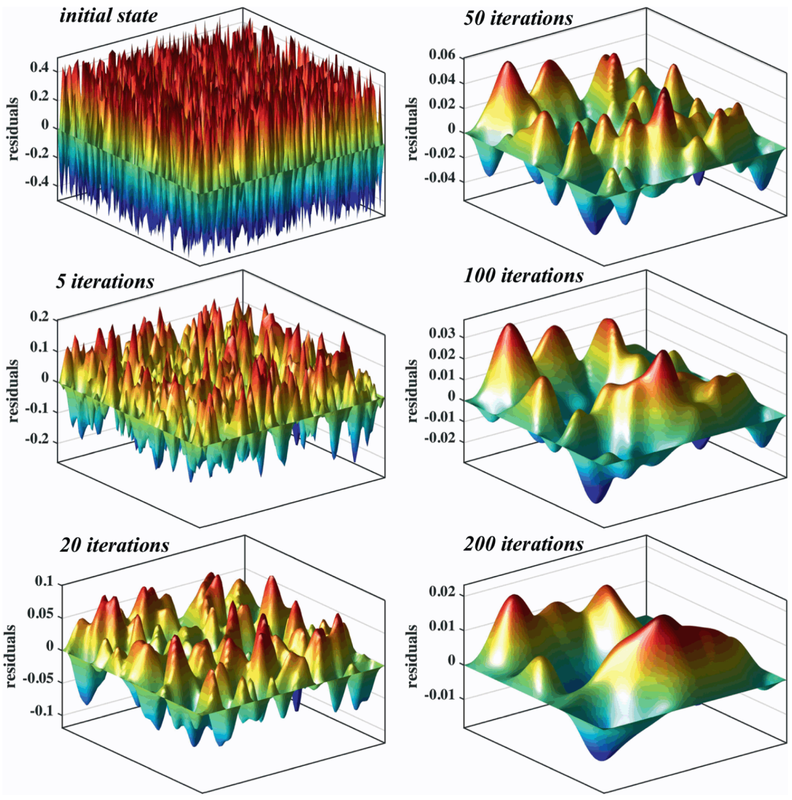


Figure 1.1: Example of Multigrid Solution in different iterations [2].

Although J.T. Oden's early works on finite element analysis date from the 1960s, they show the application of what was then a relatively poorly understood numerical method to problems that are still considered difficult by today's standards, such as large deformation elasticity, pneumatic structures, thermoelasticity, fluid flow, and incompressible elasticity [3]. The Finite Element Method (FEM) is a basic numerical method for solving partial differential equations with two or three variables in two or three dimensions. The FEM separates a big system into smaller, simpler sections called finite elements to solve a problem. The finite element method, which was first established as an ad hoc engineering procedure for obtaining stress and strain displacement solutions in structural analysis, has evolved in numerous ways [4]. The finite element method is essentially geometry-free and theoretically be used in domains of any form and with arbitrary BCs [5]. By definition, the finite element approach produces unstructured meshes where the majority of complicated geometries can be easily handled [5]. These characteristics distinguish the finite element approach as a generic, systematic, powerful, and adaptable numerical method that outperforms other numerical methods [5].

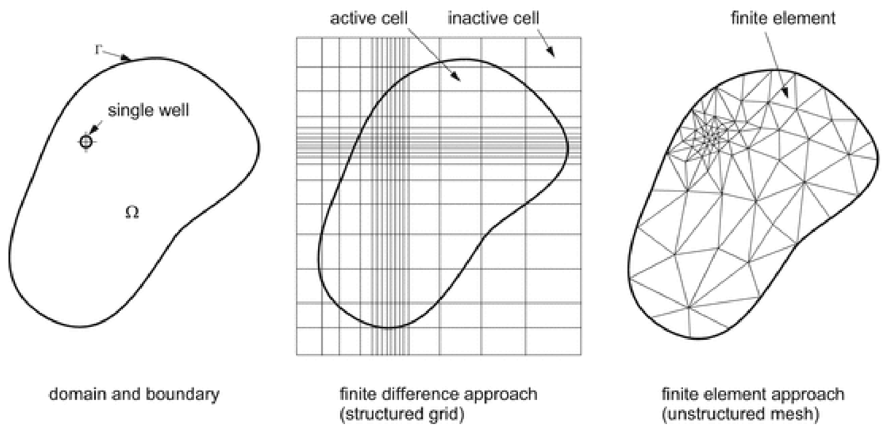

Γ
single well
Ω
active cell
inactive cell
finite element
domain and boundary
finite difference approach (structured grid)
finite element approach (unstructured mesh)

Figure 1.2: Example of 2D domain discretized by finite differences and finite elements [5].

Both these methods are combined to increase the accuracy of the solution. However, programming codes for this method are rarely found in public as the complexity of the code is very high. On the GitHub repository, there are no programming codes that implement the multigrid method with the finite element method currently. One method for implementing the finite element method with the multigrid solver is to modify the current multigrid solver code.

This project will start with finding and collecting information about the finite element method that is implemented in the multigrid solver. The GitHub repository has been used to find the application of the finite element method with a combination of multigrid solvers. The multigrid solver codes have been searched throughout this website to find the most suitable multigrid code to modify and implement the finite element method.

At the end of this project, the programming structure for the 1D Finite Element with the multigrid solver can be used to solve any simple 1D problem, such as 1D heat conduction or 1D elasticity problems. By completing this task, the performance characteristics of the multigrid solver can be identified.

1.2 Problem Statement

The Finite Element Method (FEM), combined with the multigrid (MG) method, is well accepted to yield a highly accurate solution. The MG solver is an algorithm for solving differential equations using a hierarchy of discretization. Its implementation in the FEM is however very complex, such that existing code implementations are rarely found in the public domain. For this purpose, this project will illustrate the principle of the FEM-MG solver by implementing it in a 1D problem. The task is to set up a programming structure for a 1D Finite Element with the multigrid solver that can be solved for any simple 1D heat conduction or 1D elasticity problem.

1.3 Objectives

The specific objective for this project was:

i. To implement a multigrid method for solving a Finite Element method problem.
ii. To modify the available code of Multigrid solver with Finite Element Method.
iii. To identify the performance of the solver using a different number of elements.
iv. To improve the current available Multigrid solver with Finite Element Method algorithm.

1.4 Scope of the Project

This project involves the simulation of a 1D Multigrid solver with the Finite Element Method. Different value of unknown was used for the simulated 1D problem to obtain performance characteristics. The simulation was done using Python and MATLAB software. The code was created to implement the finite element method into multigrid code by modifying an existing example code.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter will explain the finding of the research paper on the topic of a combination of the Multigrid Method with the Finite Element Method to solve the Finite Element Problem. This section also includes a deep explanation of the Multigrid Method and Finite Element Method.

2.2 Multigrid Method

The multigrid method is one of the fastest ways to solve differential equation problems. The multigrid solver is an algorithm for solving differential equations using a hierarchy of discretization. Multigrid methods are a class of algorithms used to solve computational problems [6]. It is primarily used for solving linear and non-linear boundary value problems [6]. The main reason for using the multigrid method is to accelerate the convergence of the basic iterative method. The multigrid method can be used in combination with any common discretization technique, such as the finite element method. There are many types of multigrid algorithms, but the most frequent feature used is a hierarchy of discretization. The major steps are smoothing, residual computation, restriction, interpolation, and correction. There are three main types of multigrid methods, which are V-Cycle, F-Cycle, and W-Cycle. For a discrete 2D problem, the V-Cycle iteration is the fastest type to compute, while the F-Cycle and W-Cycle iterations take 89% and 125% more time to compute, respectively. In terms of 3D problems, F-Cycle and W-Cycle take about 64% and 75% more time, respectively, compared to V-Cycle by ignoring the overheads. To reduce the complexity, this project will cover the 1D problem in combination with the finite element method.

2.3 Finite Element Method

The finite element method is a common numerical method to solve partial differential equations. This method is used widely in engineering fields, such as mathematical modelling. With the advent of digital computers, discrete problems can generally be solved readily even if the number of elements is very large [1]. As the capacity of all computers is finite, continuous problems can only be solved exactly by mathematical manipulation [1].

2.4 Combination of Multigrid Method with Finite Element Method

For the two-field issue, finite element methods are the most widely utilised numerical discretization [7]. For example, a continuous Galerkin (CG) element for both displacement and pressure are examined [7]. This paper is mainly focused on multigrid for poroelasticity problems by using the finite element method with homogeneous boundary conditions in two-dimensional space. The multigrid method is one of the most efficient iterative techniques, reducing discrete equation computation to O(N) or O(NlogN), where N is the linear system's scaling [7]. In this research, there are two mains unknown which are displacement and pore pressure. The distributive Gauss-Seidel iteration is smoother in the multigrid algorithm [7]. The demonstration of multigrid in this paper was using the V-Cycle. The result is the method does not converge within 200 iterations. The V-Cycle multigrid method with two-step pre and two-step post-smoothing is almost uniform, as the numerical results show [7]. By reviewing this paper, there is no combination of the multigrid method with the finite element method.

The computing of the solution to a system of equations is an important part of finite element techniques and iso-geometric analysis [8]. Both of these analyses were tested using the multigrid method. In this research, the finite element method was combined with an immersed method. Immersed methods are useful tools for generating body-fitted finite element discretization or analysis-suitable NURBS geometries in isogeometric analysis, especially for problems involving complex, moving, or implicitly defined geometries, to avoid time-consuming and computationally expensive procedures [8]. This is especially difficult for systems obtained via immersed techniques because such approaches typically provide system matrices that are substantially ill-conditioned [8]. The multigrid V-cycle iteration is used in these contributions. Multigrid methods successfully overcome the mesh-size dependency of linear system conditioning and its influence on iterative solution methods' convergence [8]. Although multigrid methods have not been fully investigated in the context of immersed finite element methods, there are substantial studies available on closely related features [8]. This also proves that the combination of the finite element method with multigrid is very less used. This contribution's main goal is to provide a geometric multigrid preconditioning approach that may be used with higher-order immersed finite element techniques using conventional, isogeometric, and locally refined basis functions [8]. The numerical findings are produced using a multi-grid baseline method and a reasonably simple Schwarz block selection [8]. While the computational cost scaling is already optimal in terms of system size, the structure of multigrid preconditioners with Schwarz-type smoothers allows for changes that can further improve efficiency [8].

Anomaly diffusion, material science, image processing, finance, and electromagnetic fluids all employ fractional equations to describe events [3]. In the same manner that standard diffusion equations originate from Brownian random walks, partial differential equations with fractional-order operators occur naturally when the limit of discrete diffusion is dominated by stochastic processes [3]. The effective solution of fractional equations presented on complex domains is a subject of substantial practical relevance, and computational models for the numerical resolution of models incorporating fractional derivatives in two or more dimensions are relatively sparse [3]. The goal of this project is to create all of the components needed to build an adaptive finite element algorithm for approximating fractional partial differential equations [3]. In the end, this research has expanded the fractional Laplacian operator's numerical approximation from globally quasi-uniform meshes to locally refined meshes in this paper [3]. It used posterior error indicators and adaptive mesh refinement to overcome delayed convergence caused by the inherent singularity of the solution at the boundary [3].

Fine-scale qualities impact the overall characteristics of any structural or natural material [9]. All of those micro-details, however, cannot normally be directly taken into account in modelling due to the tremendous scale and complexity of the calculation [9]. As a result, numerous homogenization approaches for a simpler analysis of heterogeneous material have been devised [9]. They studied the evolution of multigrid homogenisation in this study since it relates to materials that do not have a periodic microstructure or a scale separation feature [9]. Linear elasticity for composite materials flows in porous media, and electrostatics are examples of such problems [9]. At the macro-scale, we apply the higher-order finite element method approximation and provide a revised definition of the inter-grid operators, resulting in a rapid convergence of both displacements and stresses in terms of not only the number of degrees of freedom but also the CPU time [9]. This study used a new approach of inter-grid mapping building to apply the hierarchical approximation of order up to five for multigrid homogenization [9]. The numerical studies reveal a quick reduction in the modelling error that each homogenization method invariably introduces [9].

Direct numerical simulation of solid-liquid two-phase flows is challenging due to the uneven domain inhabited by the fluid, which varies with particle motion [10]. Furthermore, because the particles are advected by the fluid and exert forces on it, the body-liquid interaction necessitates the calculation of the fluid stress at the fluid-solid interface, especially in the case of large numbers of particles [10]. Besides, the interactions between the fluid and the particles, as well as particle collisions, add to the problem's complexity [10]. To overcome such a problem, two different ways have been proposed. The first is an extended ALE standard Galerkin finite element method that incorporates both the fluid and particle equations of motion into a single linked variational equation [10]. The second method is based on the concept of fictitious or embedded domains [10]. The research has proposed a multigrid finite element method based explicit fictitious boundary method [10]. A multigrid finite element solver calculates the flow, and solid particles are free to travel through the computational mesh, which may be selected separately from particles of any form, size, or number [10]. The multigrid finite element method fictitious boundary technique has been introduced for the direct numerical modelling of solid-liquid two-phase flows in 2D with many moving particles [10]. Several comparisons between the reported findings and relevant reference results from our calculations or the literature have demonstrated the reliability of the method [10].

Several engineering situations are modelled using partial differential equations (PDEs) [11]. These equations must adhere to the specific boundary and initial conditions, creating an analytical solution a difficult and not always doable process [11]. This paper provides a model for solving the hyperbolic PDE which is the wave equation [11]. The finite difference method discretizes the transient wave equation, resulting in a linear system that may be solved using a solver such as the Gauss-Seidel method [11]. Applying the multigrid algorithm, which is highly advocated in the literature and considerably enhances the convergence factor, is one way to speed up the process of getting the solution to this system [11]. The time-stepping method may be used to solve physical problems described by PDEs that are time-dependent, in which the solution of the previous time step is used as an initial estimate at each subsequent time step, therefore solving a transient problem with a sequence of stationary wave equations [11]. To solve the wave equation discretized using the finite difference method, the methodology provided here uses time-stepping with the multigrid method [11]. When external forces acting on the problem are considered, similar results with the same efficiency can be achieved [11]. There are various studies in the literature that demonstrate the effectiveness of using the time-stepping technique in combination with multigrid to solve EDPs [11]. Finally, this paper has described a system for solving one and two-dimensional wave propagation problems using the finite difference technique and weighted by a parameter η at various time stages [11]. This research has discovered that by combining the multigrid algorithm with the lexicographical Gauss-Seidel solver to solve the resultant system of equations, we were able to get discretization errors that were close to our expectations while reducing computing time and linear complexity [11].

2.5 Conclusion of the Research

In conclusion, the combination of the multigrid method and the finite element method is very rare in public research. There are fewer papers that mention this combination with the 1D problem. Therefore, the best solution to this problem can be proposed in this project.

This project is proposed to provide a solution to the finite element method problem by using the multigrid solver. The multigrid solver is widely used in most programming software to solve differential equations. The multigrid solver can be implemented in the finite element method to increase the accuracy of the solution. However, the combination of this method is very complex, such that existing code implementations are rarely found in the public. Even very famous software such as ANSYS also has limited uses for the multigrid method. This shows that it is not easy to implement the finite element method into the multigrid method. However, this project will measure the performance of the current multigrid solver in MATLAB.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Overview

This chapter will go through the process that has been performed to complete this project. The process started with finding a programming code followed by a performance test and lastly modifying the code to increase the performance.

The programming code must be able to solve the matrix equation in order to solve the 1D finite element problem such as the deflection of the cantilever beam, the heat conduction problem and the elasticity problem. The matrix equation is shown below.

$$A \cdot X = B$$

where,

$A =$ A constant value of the problem. (For example the position on the cantilever beam)

$X =$ An unknown that needs to be found. (For example the deflection of the cantilever beam)

$B =$ A variable function. (For example the function of force moment on the cantilever beam)

3.2 1-D Case Study: Cantilever Beam Deflection

A cantilever beam is a rigid structural part that is horizontally laid and supported by only one static end. The type of cantilever that is used in this problem is the moment connection on the wall under uniform load distributed. The general formula for the deflection of a cantilever beam under a uniformly distributed load is,

$$\theta = \frac{\omega l^3}{6EI}$$

where,

$\theta =$ Deflection of the beam

$\omega =$ Load applied on the beam

$l =$ The length of the beam

$E =$ Young’s Modulus

$I =$ Moment of Inertia

The problem statement of this case study is the deflection ‘u’ of a cantilever beam under uniformly distributed load is governed by,

$$\frac{d^2u}{dx^2} = -150 + 300x - 150x^2$$

The boundary conditions are:

$$u = \frac{\partial u}{\partial x} = 0 \qquad at \quad x = 0$$

Plot the beam deflection as a function of length. Take the length of the beam as 5 units. Discretize the governing equations using FEM and solve using the Multigrid Gauss-Seidel method [12].

The discretization of the equation is using Galerkin’s Method with the final form of the equation being in a matrix equation, KX=F, [12]

$$\frac{1}{\Delta x}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} u_i \\ u_j \end{bmatrix} =$$

$$\begin{bmatrix} \frac{-du_i}{dx} \\ \frac{du_j}{dx} \end{bmatrix} + \frac{150}{\Delta x}\begin{bmatrix} \left(\frac{x_j^4 + 3x_i^4}{12}\right) - \left(\frac{x_j^3 + 2x_i^3}{3}\right) + \left(\frac{x_j^2 + x_i^2}{2}\right) - x_j\left(\frac{x_i^3}{3} - x_i^2 + x_i\right) \\ \left(\frac{x_i^4 + 3x_j^4}{12}\right) - \left(\frac{x_i^3 + 2x_j^3}{3}\right) + \left(\frac{x_i^2 + x_j^2}{2}\right) - x_i\left(\frac{x_j^3}{3} - x_j^2 + x_j\right) \end{bmatrix}$$

3.3 Finding of MG Method Coding with the Implementation of FEM

This project starts with finding the most suitable coding that includes both MG and FEM in the algorithm. The finding was made through the GitHub repository. GitHub is a code hosting platform for version control and collaboration. Many developers will use this repository to publish their code of the program for other users to use. This platform is used to find any suitable algorithm for this project. There are lots of results found by searching in this repository but it is only a few algorithms that provide the combination of other methods with the MG method. At last, a set of programming codes using MATLAB software was found that implement FEM in the MG method. This programming code was published two years ago by Nitish Gudapati with GitHub user ID “gnitish18”. This code is used Multigrid Gauss-Seidel to solve a finite element analysis of a cantilever beam. There are six MATLAB codes provided by the owner which are one main code and five function codes.

The list of the files is:

i. FEM_Multigrid.m

ii. Animate.m

iii. Multigrid_TwoGrid.m

iv. Gauss_Seidel.m

v. Project_FineCoarse.m

vi. Result.m

All six files are important in order to perform the solution of the deflection problem of the cantilever beam. The FEM_Multigrid.m file is the main body of the program that consists of other five functions in it. The Animate.m file is a function to display the animation of beam deflection. The Multigrid_TwoGrid.m file is a Function to solve the matrix equation using the Multigrid Algorithm implementing two-grid. This function is the most important because, in this function where happen all calculations to solve the value of X. This function, it consists of two other functions which are the “Gauss_Seidel” function and “Project_FineToCoarse” function. The Gauss_Seidel.m file is a function to solve the matrix equation using the iterative Gauss-Seidel relaxation method. The Project_FineToCoarse.m file is a function to project a given Fine-Grid matrix to Coarse-Grid. Lastly, the Results.m file is a function to plot the analytical solution and compare it with the FEM solution.

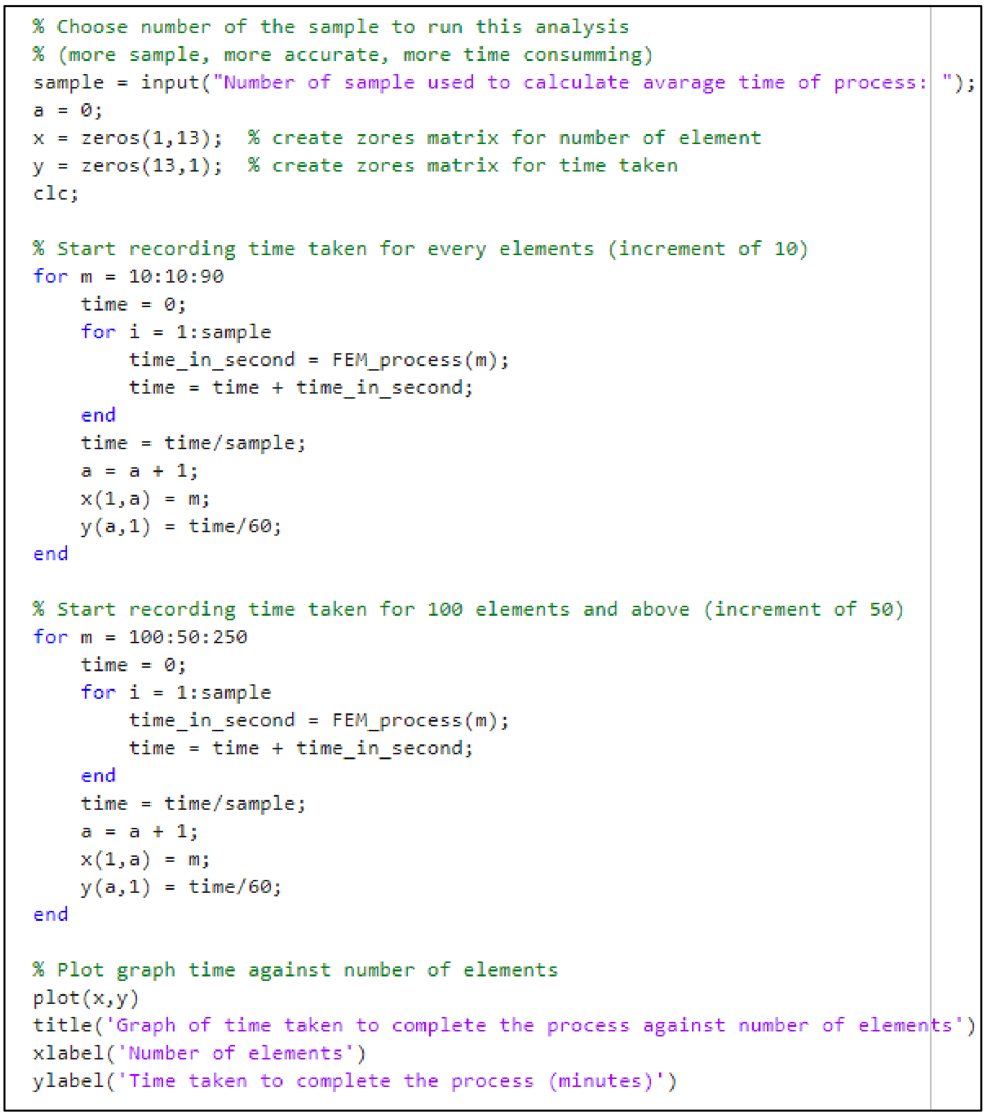
3.4 Hardware and Software

This project was a simulation and programming-type project. The hardware used in this project to run the simulation is a Windows 11 laptop with AMD Ryzen 7 4800H and 16GB of installed RAM. The software used for the simulation is MATLAB programming software. The software was installed with the basic add-on package.

3.5 Performance Test of Algorithm

Before performing the performance test, the algorithm was executed to make sure there is no errors occurred while running the code. The process starts by choosing a number of elements that need to be used for the calculation. The higher number of elements, the higher accuracy of the result.

The process was continued by determining the parameter used to record the performance. The parameter used in this test is the time taken to complete the solver process. The executing time was recorded for every number of elements starting with 10 elements up to 250 elements. In order to plot a graph of time taken against the number of elements, one new main coding was created and the old main coding was changed to the FEM function coding. The “tictoc” function was added to the FEM function to record the time taken. The new main program to record the execution time of the process is shown in Figure 3.1 below.

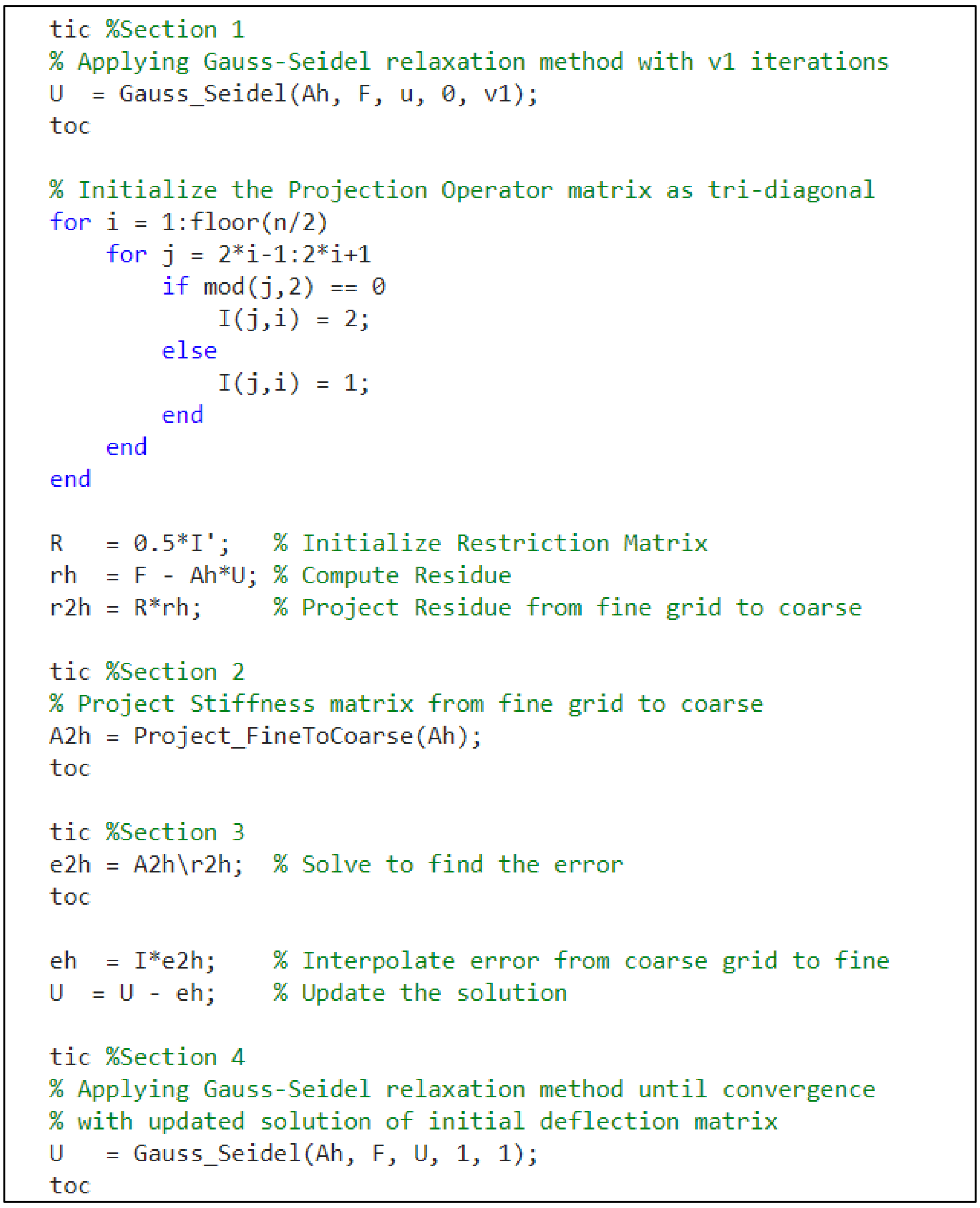


```matlab
% Choose number of the sample to run this analysis
% (more sample, more accurate, more time consumming)
sample = input("Number of sample used to calculate avarage time of process: ");
a = 0;
x = zeros(1,13);  % create zores matrix for number of element
y = zeros(13,1);  % create zores matrix for time taken
clc;

% Start recording time taken for every elements (increment of 10)
for m = 10:10:90
    time = 0;
    for i = 1:sample
        time_in_second = FEM_process(m);
        time = time + time_in_second;
    end
    time = time/sample;
    a = a + 1;
    x(1,a) = m;
    y(a,1) = time/60;
end

% Start recording time taken for 100 elements and above (increment of 50)
for m = 100:50:250
    time = 0;
    for i = 1:sample
        time_in_second = FEM_process(m);
        time = time + time_in_second;
    end
    time = time/sample;
    a = a + 1;
    x(1,a) = m;
    y(a,1) = time/60;
end

% Plot graph time against number of elements
plot(x,y)
title('Graph of time taken to complete the process against number of elements')
xlabel('Number of elements')
ylabel('Time taken to complete the process (minutes)')
```

Figure 3.1: The main body program to record the time taken

3.6 Problem Detection

To improve the performance of the process, the problem needs to be searched in the coding. The main problem that needs to be searched is which line took a long time to process. Four sections of the coding in the “Multigrid_TwoGrid” function file were decided to be tested that potentially slowing down the process. All sections were added with the “tictoc” function to record the time taken to run on every section. The sections of the coding that were tested are shown in Figure 3.2 below.



```
tic %Section 1
% Applying Gauss-Seidel relaxation method with v1 iterations
U  = Gauss_Seidel(Ah, F, u, 0, v1);
toc

% Initialize the Projection Operator matrix as tri-diagonal
for i = 1:floor(n/2)
    for j = 2*i-1:2*i+1
        if mod(j,2) == 0
            I(j,i) = 2;
        else
            I(j,i) = 1;
        end
    end
end

R   = 0.5*I';   % Initialize Restriction Matrix
rh  = F - Ah*U; % Compute Residue
r2h = R*rh;     % Project Residue from fine grid to coarse

tic %Section 2
% Project Stiffness matrix from fine grid to coarse
A2h = Project_FineToCoarse(Ah);
toc

tic %Section 3
e2h = A2h\r2h;  % Solve to find the error
toc

eh  = I*e2h;    % Interpolate error from coarse grid to fine
U  = U - eh;    % Update the solution

tic %Section 4
% Applying Gauss-Seidel relaxation method until convergence
% with updated solution of initial deflection matrix
U   = Gauss_Seidel(Ah, F, U, 1, 1);
toc
```

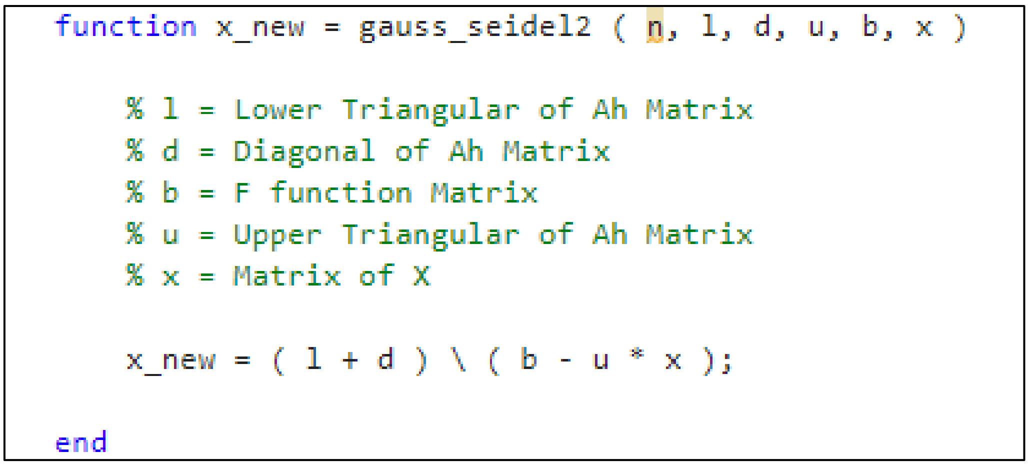
Figure 3.2: The coding of four sections was tested

In section 1, the Gauss-Seidel relaxation method is used to calculate the value of X. Same goes with section 4, which also used the Gauss-Seidel relaxation method to solve the value of X. These sections are suspected to be slowing down the process because of "Gauss_Seidel" function has the "while" loop and lots of the "for" loop. For section 2, even though the coding of the "Project_FineCoarse" function is short, it also consists of a few loop functions that might be slowing down the process. Lastly, section 3 was chosen to be tested because this part is using the back-slash function that also can take some time solving it.

The test was run three times using three different values of elements which are 50, 100 and 150 elements. The result of the test shows that section 4 is the slowest in terms of time compared to other sections. Even though section 1 also used the same function, section 4 was applying the Gauss-Seidel relaxation method until convergence which took more time to complete the calculation.

3.7 Coding Modification

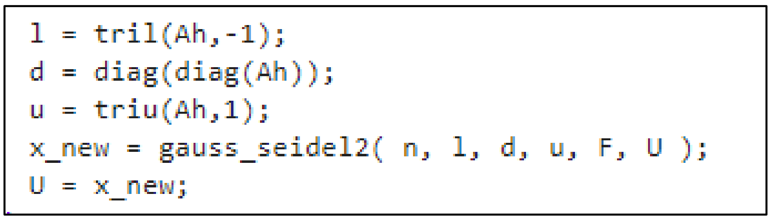
After identifying the problem section, the slowest section in time is modified to a new coding function. The new Gauss-Seidel function was created to replace the "Gauss_Seidel" function in section 4. The new Gauss-Seidel function was named as "gauss_seidel2" function. This function was provided by the author [13] by referring to one of the paper's authors C.T. Kelley [14]. The function is different from the old Gauss-Seidel function. This function implemented the mathematical symbol of the back-slash operation to calculate the value of X. the equation consists of matrix B, matrix X, upper and lower triangular matrix of matrix A and diagonal matrix of matrix A. The coding of the "gauss_seidel2" function was modified to simpler codes as shown in Figure 3.3 below.



```
function x_new = gauss_seidel2 ( n, l, d, u, b, x )

    % l = Lower Triangular of Ah Matrix
    % d = Diagonal of Ah Matrix
    % b = F function Matrix
    % u = Upper Triangular of Ah Matrix
    % x = Matrix of X

    x_new = ( l + d ) \ ( b - u * x );

end
```

Figure 3.3: New function of the Gauss-Seidel Method

Next, the section 4 code also needs to be changed. First, the Ah matrix needs to be split into three matrices which are upper triangular, lower triangular and diagonal. In order to split those matrices, the "tril" function, "triu" function and "diag" function were used. The new coding of section 4 was shown in Figure 3.4 below.



```
l = tril(Ah,-1);
d = diag(diag(Ah));
u = triu(Ah,1);
x_new = gauss_seidel2( n, l, d, u, F, U );
U = x_new;
```

Figure 3.4: New coding for section 4

Finally, the main coding was modified to make it more clean and simple. The maximum number of elements also increases to 500 elements. Then, the final code was tested again using 250 elements to compare with the previous code performance. In addition, the final code was also tested using the maximum number of elements and the graph of time against the number of elements was plotted. The overall coding was shown in the appendices section.

3.8 Flowchart of the Algorithm

This sub-chapter will provide the overall flow of the process in form of a flowchart. There are 3 three main flowcharts that will be shown in this part which are the main body process, the FEM function and the Multigrid_TwoGrid function. The flowchart of the main program, FEM function and Multigrid_TwoGrid function was shown in Figure 3.5, Figure 3.6 and Figure 3.7 respectively. This flowchart will help users easily understand the flow of the program.

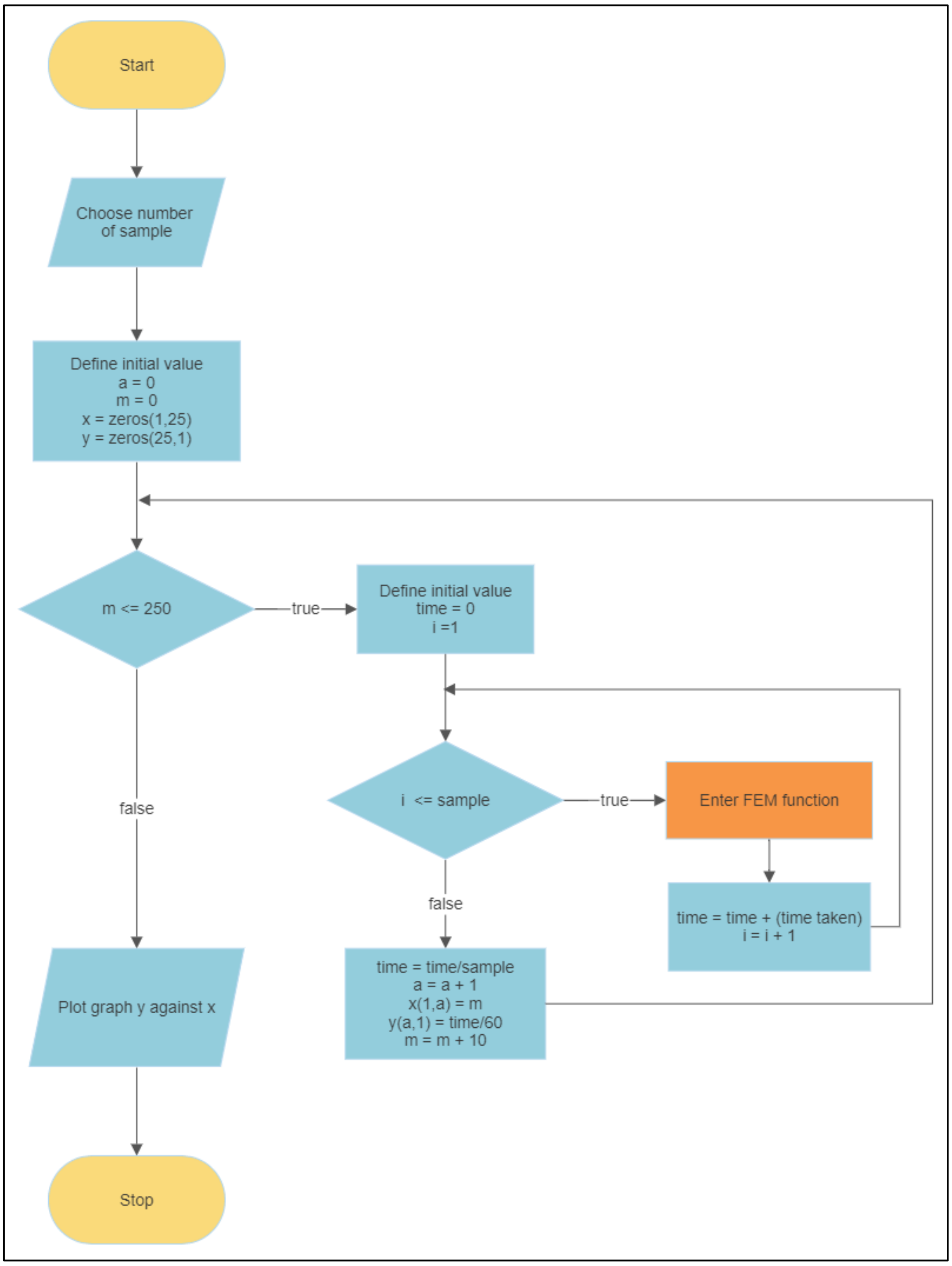


Figure 3.5: Flowchart of the main program

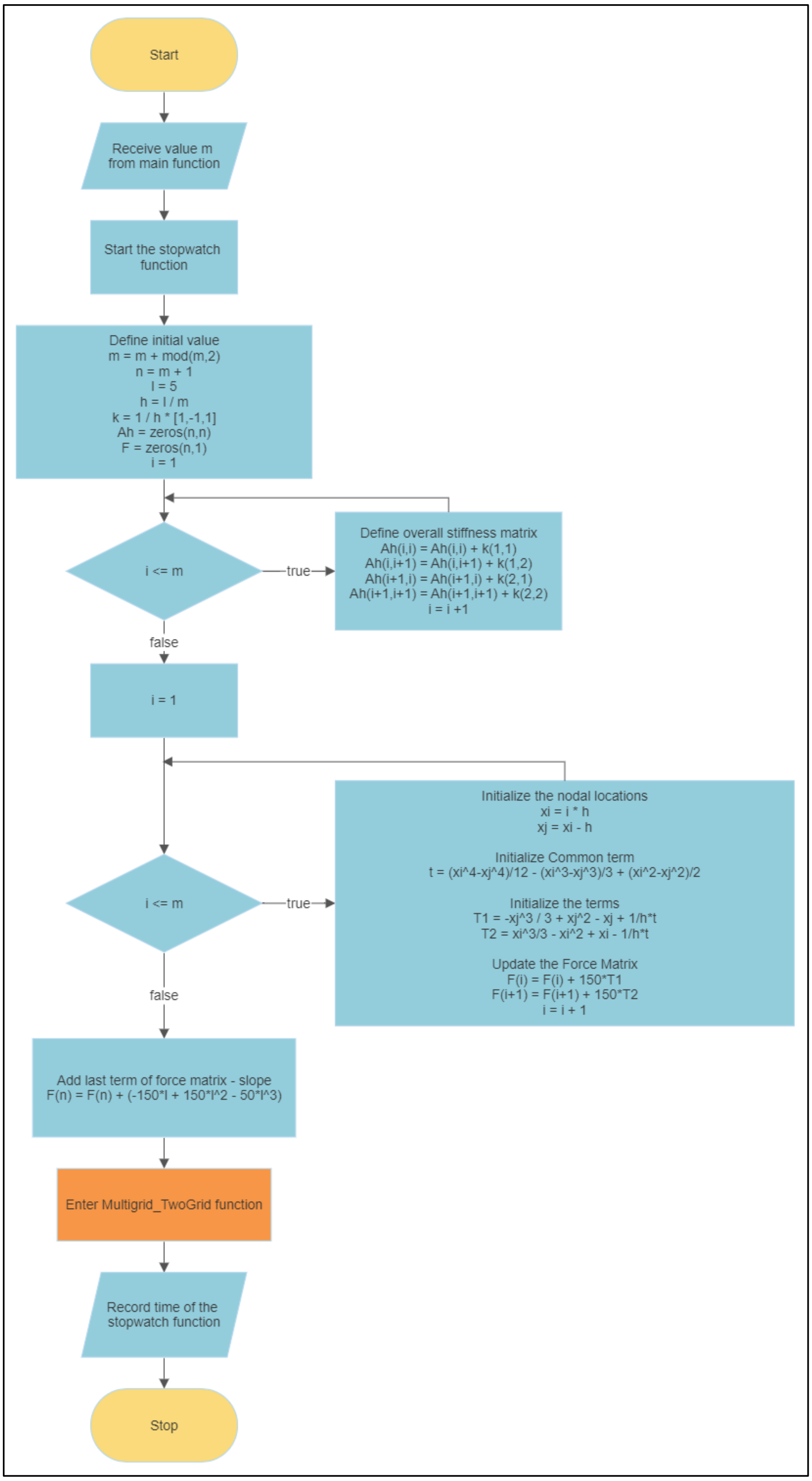

Start
Receive value m from main function
Start the stopwatch function
Define initial value
m = m + mod(m,2)
n = m + 1
l = 5
h = l / m
k = 1 / h * [1,-1,1]
Ah = zeros(n,n)
F = zeros(n,1)
i = 1
i <= m
true
Define overall stiffness matrix
Ah(i,i) = Ah(i,i) + k(1,1)
Ah(i,i+1) = Ah(i,i+1) + k(1,2)
Ah(i+1,i) = Ah(i+1,i) + k(2,1)
Ah(i+1,i+1) = Ah(i+1,i+1) + k(2,2)
i = i +1
false
i = 1
i <= m
true
Initialize the nodal locations
xi = i * h
xj = xi - h
Initialize Common term
t = (xi^4-xj^4)/12 - (xi^3-xj^3)/3 + (xi^2-xj^2)/2
Initialize the terms
T1 = -xj^3 / 3 + xj^2 - xj + 1/h*t
T2 = xi^3/3 - xi^2 + xi - 1/h*t
Update the Force Matrix
F(i) = F(i) + 150*T1
F(i+1) = F(i+1) + 150*T2
i = i + 1
false
Add last term of force matrix - slope
F(n) = F(n) + (-150*l + 150*l^2 - 50*l^3)
Enter Multigrid_TwoGrid function
Record time of the stopwatch function
Stop

Figure 3.6: Flowchart of the FEM function

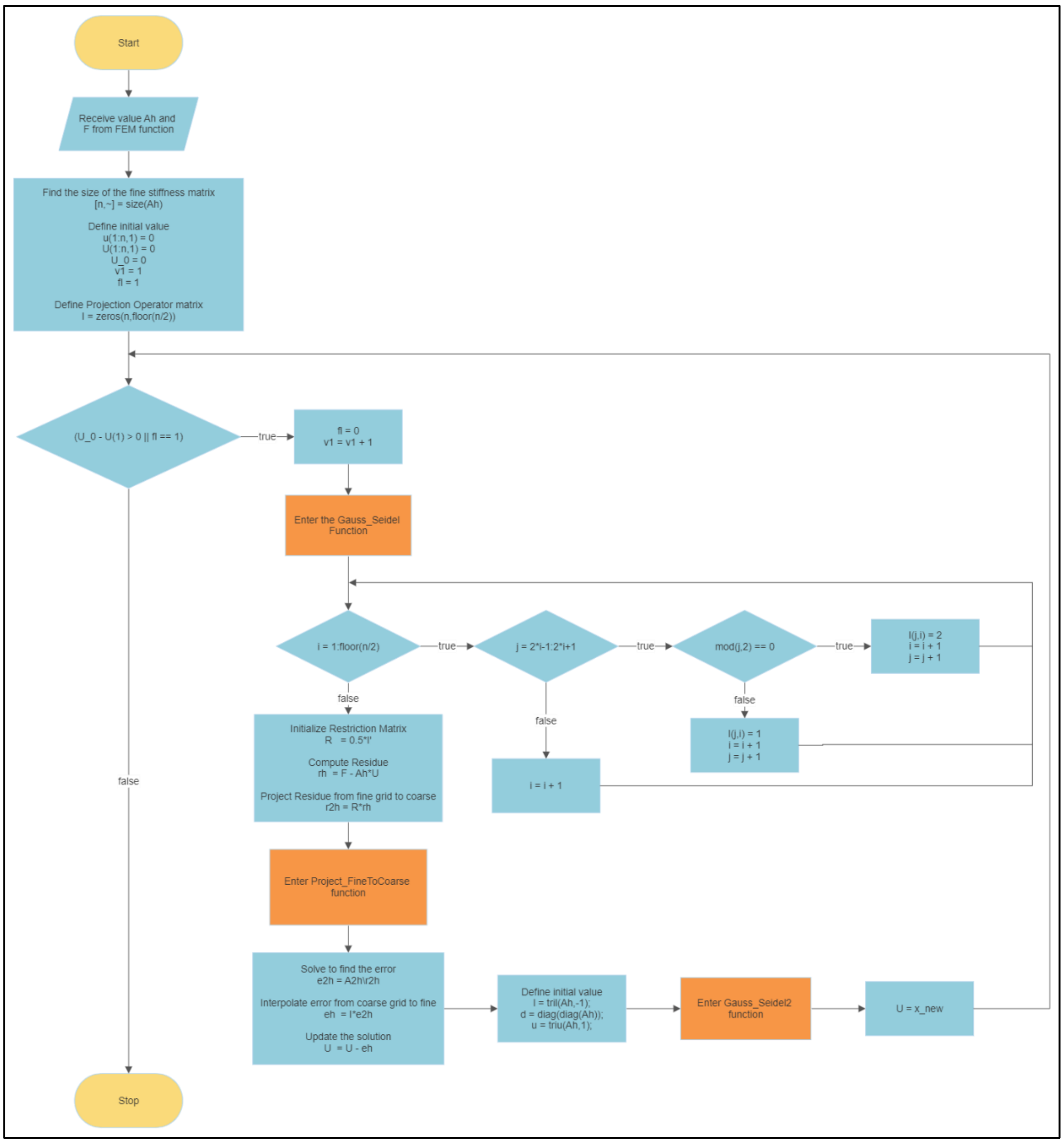


Figure 3.7: Flowchart for Multigrid_TwoGrid function

Based on Figure 3.5, the process of the main program is a sample and easy to understand. In this process, there is a loop function containing the FEM function which is in an orange colour box. These orange boxes indicate the process is entering another flowchart process. Therefore, the main program also contains one other function.

Based on Figure 3.6, the flowchart shows that there is a lot of complex calculation and one other function in the process. In this process, it focused on obtaining the initial value. At the end of the process, it will return to the main program to record execution time.

Lastly, Figure 3.7 shows all the calculation processes to produce the deflection result. This process consists of three other short functions which are the Gauss_Seidel function, Project_FineCoarse function and Gauss_Seidel2 function. At the end of this flow will return to the FEM function process.

3.9 Coding Explanation

This MG method solver with FEM algorithm consists of six MATLAB files which are one main program file and five function files. The explanation of the main program, FEM function, Multigrid_TwoGrid function, Gauss_Seidel function, Project_FineCoarse Function and Gauss_Seidel2 function codes as shown in Table 3.1, Table 3.2, Table 3.3, Table 3.4, Table 3.5 and Table 3.6 respectively.

Table 3.1: The main program code explanation

MATLAB codes	Description

`clear;`
`close all;`
`clc;`
	Clear all workspace values, close all windows and a clear command window.

`sample = input("Number of sample: ");`	Choose the number of samples to run this analysis (more samples, more accurate, more time-consuming).

`a = 0;`
`x = zeros(1,50);`
`y = zeros(50,1);`
	Define the initial value and create a zores matrix for the number of elements and time taken.

`for m = 10:10:500`
`    time = 0;`
`    for i = 1:sample`
`        time_in_second = FEM_process(m);`
`        time = time + time_in_second;`
`    end`
`    time = time/sample;`
`    a = a + 1;`
`    x(1,a) = m;`
`    y(a,1) = time/60;`
`end`
	Start recording the time taken for every element up to 500 elements (increment of 10).

`plot(x,y)`
`title('Graph of time taken to complete the process against number of elements')`
	Plot graph time against the number of elements.

`xlabel('Number of elements')`
`ylabel('Time taken to complete the process (minutes)')`	

Table 3.2: The FEM function code explanation

MATLAB codes	Description
`function time_in_second = FEM_process(m)`	Define function.
`tic`	Start the stopwatch function.
`m = m + mod(m,2);`	Change the odd number of elements to even in order to satisfy the multigrid transformation matrix sizes.

`n      = m+1;`
`l      = 5;`
`h      = l/m;`
`k      = 1/h*[1, -1; -1, 1];`
`Ah     = zeros(n,n);`
`F      = zeros(n,1);`
	Define the initial value:
	i. Initialize the number of nodes
	ii. Define the length of the beam
	iii. Length of each element
	iv. Individual stiffness matrix
	v. Deflection matrix
	vi. Force matrix

`for i = 1:m`
`Ah(i,i)     = Ah(i,i)     + k(1,1);`
`Ah(i,i+1)   = Ah(i,i+1)   + k(1,2);`
`Ah(i+1,i)   = Ah(i+1,i)   + k(2,1);`
`Ah(i+1,i+1) = Ah(i+1,i+1) + k(2,2);`
`end`
	For loop to define the overall stiffness matrix.

`for i = 1:m`
`xi = i*h;`
`xj = xi-h;`
 	
`t = (xi^4-xj^4)/12 - (xi^3-xj^3)/3 + (xi^2-xj^2)/2;`
 	
`T1 = -xj^3/3 + xj^2 - xj + 1/h*t;`
`T2 = xi^3/3 - xi^2 + xi - 1/h*t;`
 	
`F(i)   = F(i)   + 150*T1;`
`F(i+1) = F(i+1) + 150*T2;`
`end`
	For loop to form force matrix (F) in A*X = F.
	The operation is:
	i. Initialize the nodal locations
	ii. Initialize Common term
	iii. Initialize the terms
	iv. Update the Force Matrix

`F(n) = F(n) + (-150*l + 150*l^2 - 50*l^3);`	Add the last term of the force matrix – slope
`U = Multigrid_TwoGrid(Ah, F);`	Solve the equation using Multigrid Gauss-Seidel.
`time_in_second = toc;`	Record the time of the stopwatch and return to the main program.

Table 3.3: The Multigrid_TwoGrid function explanation

MATLAB codes	**Description**
`function U = Multigrid_TwoGrid(Ah, F)`	Define function.
`[n,~]     = size(Ah);`	Determine the size of the matrix.

`u(1:n,1) = 0;`
`U(1:n,1) = 0;`
`U_0      = 0;`
`v1       = 1;`
`fl       = 1;`
`I = zeros(n,floor(n/2));`
	Define the initial value of:
	i. Initialization of Initial Deflection matrix.
	ii. Define the Final Deflection matrix.
	iii. Boundary condition.
	iv. Number of iterations for Gauss-Seidel.
	v. Flag to ensure the first iteration.
	vi. Define Projection Operator matrix.

`while (U_0 - U(1) > 0 || fl == 1)`	While loop with condition $U_0 - U_1$ larger than 0 or flag value equal to 1. The flag condition is used to make sure the first loop is run.

`fl = 0;`
`v1 = v1 + 1;`
	Disable the flag and increase the number of iterations.

`U  = Gauss_Seidel(Ah, F, u, 0, v1);`	Applying the Gauss-Seidel relaxation method with v1 iterations.

`for i = 1:floor(n/2)`
`   for j = 2*i-1:2*i+1`
`       if mod(j,2) == 0`
`            I(j,i) = 2;`
`       else`
`            I(j,i) = 1;`
`       end`
`    end`
`end`
	Initialize the Projection Operator matrix as tri-diagonal.

`R   = 0.5*I';`	Initialize the Restriction Matrix.
`rh  = F - Ah*U;`	Compute Residue.
`r2h = R*rh;`	Project Residue from the fine grid to coarse.
`A2h = Project_FineToCoarse(Ah);`	Project Stiffness matrix from the fine grid to coarse.
`e2h = A2h\r2h;`	Solve to find the error.
`eh  = I*e2h;`	Interpolate error from a coarse grid to fine.
`U  = U - eh;`	Update the solution

`l = tril(Ah,-1);`
`d = diag(diag(Ah));`
`u = triu(Ah,1);`
	Define value for upper triangular, lower triangular and diagonal matrix for Ah.

`x_new = gauss_seidel2( n, l, d, u, F, U );`	Applying the new Gauss-Seidel method.
`U = x_new;`	Enter the new value of X into U.

Table 3.4: The Gauss-Seidel function explanation

MATLAB codes	**Description**
`function X = Gauss_Seidel(A, B, X, fl, v)`	Define function.
`if ~fl`	The decision function if the flag is disabled, runs the first part.
`for k = 1:v`	Run the loop until the k value is equal to the number of iterations.
`x_old = X;`	Store previous iteration values.