

**An-Najah National University**

**Faculty of Graduate Studies**

# **Numerical Methods for Solving Hyperbolic Type Problems**

**By**

**Anwar Jamal Mohammad Abd Al-Haq**

**Supervisor**

**Prof. Naji Qatanani**

**This Thesis is Submitted in Partial Fulfillment of the Requirements for  
the Degree of Master of Computational Mathematics, Faculty of  
Graduate Studies, An-Najah National University, Nablus, Palestine.**

**2017**

# **Numerical Methods for Solving Hyperbolic Type Problems**

**By**

**Anwar Jamal Mohammad Abd Al-Haq**

**This thesis was defended successfully on 29/3 /2017 and approved by :**

**Defense Committee Members**

**Signature**

– **Prof. Naji Qatanani /Supervisor**

.....

– **Dr. Mahmoud Al- Manassra /External Examiner**

.....

– **Dr. Hadi Hamad /Internal Examiner**

.....

### III

## **Dedication**

I dedicate my work to all my family members, to my parents, my love my fiancé Ahmad, my sisters, my brothers who encourage me to learn, grow and develop and who have been a source of encouragement and inspiration to me.

## **Acknowledgement**

In the beginning, I am grateful to God to complete this thesis. I wish to express my sincere thanks to Prof. Dr. Naji Qatanani for providing me with all the necessary facilities for the research and I am extremely thankful and indebted to him for sharing expertise, sincere valuable guidance and encouragement extended to me.

My thanks also external examiner Dr. Mahmoud Al- Manassra and to my internal examiner Dr. Hadi Hamad for their useful and valuable comments. Also, my great thanks are due to my family and my fiance Ahmad for their support, encouragement and great efforts for me.

My sense of gratitude to everyone, who directly or indirectly, have lent their hand in this venture.

## الإقرار

انا الموقع أدناه مقدم الرسالة التي تحمل عنوان :

# **Numerical Methods for Solving Hyperbolic Type Problems**

أقر بأن ما اشتملت عليه هذه الرسالة انما هي نتاج جهدي الخاص، باستثناء ما تمت الإشارة اليه  
حيثما ورد، وأن هذه الرسالة ككل، أو أي جزء منها لم يقدم من قبل لنيل أي درجة علمية أو بحث  
علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

## **Declaration**

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

**Student's name:**

**اسم الطالب:**

**Signature:**

**التوقيع:**

**Date:**

**التاريخ:**

## Table of Content

Dedication .....	III
Acknowledgement.....	IV
Declaration .....	V
Table of Content.....	VI
List of Figures .....	VIII
List of Tables.....	IX
Abstract .....	X
Introduction .....	1
Chapter One.....	5
Finite Difference and Finite Element Methods for Solving Hyperbolic Partial Differential Equations.....	5
1.1 Hyperbolic PDE Subject to Boundary Conditions .....	6
1.2 Discretization of Hyperbolic PDE by Finite Difference Method .....	6
1.3 The Principle of Finite Difference Method (FDM) .....	6
1.4 Strategy of Discretization .....	7
1.4.1 Homogeneous Wave Equation with Dirichlet Boundary Conditions .....	12
1.4.2 Inhomogeneous Wave Equation with Dirichlet Boundary Conditions .....	17
1.4.3 Homogeneous Wave Equation with Neumann Boundary Conditions .....	17
1.4.4 Inhomogeneous Wave Equation with Neumann Boundary Conditions .....	19
1.5 Finite Element Method .....	20
The Principle of Finite Element Method (FEM) .....	21
1.5.1 Finite Element Method (FEM) for Dirichlet Boundary Value Problems .....	22
1.5.2 Finite Element Method (FEM) with Neumann Boundary .....	29
1.6 Finite Difference Method for Two Dimensional Wave Equation ..	33
1.6.2 Two Dimensional Wave Equation with Neumann Boundary Conditions .....	41
Chapter Two.....	44
Iterative Methods for Solving Linear Systems .....	44
2.1 Jacobi Method .....	46
2.2 Gauss-Seidel Method.....	48

## VII

2.3 Successive over Relaxation Method (SOR Method) .....	49
2.4 Conjugate Gradient Method.....	52
2.5 Convergence of Iterative Methods.....	54
2.5.1 Convergence of Jacobi and Gauss-Seidel Iterative Methods .....	55
2.5.2 Convergence of SOR iterative Method.....	56
2.5.3 Convergence of Conjugate Gradient Method .....	56
Chapter Three.....	59
Numerical Results .....	59
3.1 Comparison between results for finite difference method and finite element method for example 3.1:.....	97
3.2 Comparison between results for iterative methods.....	97
3.3 Conclusions .....	98
References .....	99
Appendix A .....	105
Appendix B .....	107
Appendix C .....	109
Appendix D .....	111
Appendix E.....	114
Appendix F.....	116
Appendix G .....	118
Appendix H .....	120
Appendix I.....	123
Appendix J.....	125
Appendix K .....	127
Appendix L.....	129
الملخص .....	ب

## List of Figures

Figure1. 1: discretization of a rectangular domain .....	8
Figure1. 2: 5-points stencil of the unknown function $u(x, t)$ .....	11
Figure1. 3: three points $i + 1, i$ , and $i - 1$ which are located on $x$ -axis..	13
Figure 1. 4: three points $j + 1, j$ , and $j - 1$ which are located on the $y$ -axis .....	14
Figure1. 5: combining x-axis and t-axis around the $(i, j)$ point .....	15
Figure 1. 6: Dirichlet boundary conditions defined on the rectangular domain .....	16
Figure1. 7: finite element on irregular shape .....	22
Figure1. 8: rectangular domain with Dirichlet boundary conditions.....	23
Figure1. 9: coordinate for each node in finite element method .....	24
Figure1. 10: the local node numbers are determined on nodes start from node1, then node 2 and finally with node 10 (in a counterclockwise).....	25
Figure1. 11: Node 1 is assigned local node number 1 in element 1 .....	27
Figure1. 12: Node 2 has Local node number 2 in element 1 and Local node number 1 in element 2 and element 3.....	27
Figure1. 13: discretization of two dimensional domain .....	34
Figure3. 1: discretization of the domain for example 1 .....	61
Figure3. 2: discretization the domain with dirichlet boundary condition for example 1.....	62
Figure 3. 3: discretization the domain by finite element method for example 2 .....	73
Figure3. 4: discretization the domain with Neumann boundary condition for example 3 .....	82
Figure3. 5: discretization the domain with Dirichlet boundary conditions for example 3 .....	90



## List of Tables

Table 3. 1: Comparison between the iterative methods for example 1 .....	72
Table 3. 2: the global nodes, local node numbers and the coordinates for each element.....	74
Table 3. 3: represents vector of prescribed boundary values .....	79
Table 3. 4: matrix $\mathbf{Cvv}$ that obtained from global coefficient matrix $\mathbf{C}$ ....	80
Table 3. 5: matrix $\mathbf{Cvn}$ that obtained from global coefficient matrix $\mathbf{C}$ .....	80
Table 3. 6: Comparison between the iterative methods for example2 .....	88
Table 3. 7: Comparison between the iterative methods for example 3 .....	96
Table 3. 8: Comparison between results for finite difference method and finite element method for example 1 .....	97
Table 3. 9: Comparison between FE and FD solutions .....	97

X  
**Numerical Methods for Solving Hyperbolic Type Problems**  
**By**  
**Anwar Jamal Mohammad Abd Al-Haq**  
**Supervisor**  
**Prof. Naji Qatanani**

**Abstract**

Hyperbolic Partial Differential Equations play a very important role in science, technology and arise very frequently in physical applications as models of waves. Hyperbolic linear partial differential equations of second order like wave equations are the ones to be considered. In fact, most of these physical problems are very difficult to solve analytically. Instead, they can be solved numerically using some computational methods .

In this thesis, homogeneous and inhomogeneous wave equations with different types of boundary conditions will be solved numerically using the finite difference method (FDM) and the finite element method (FEM) to approximate the analytical (exact) solution of hyperbolic PDEs. The discretizing procedure transforms the boundary value problem into a linear system of  $n$  algebraic equations that can be solved by iterative methods. These iterative methods are: Jacobi, Gauss-Seidel, SOR, and Conjugate Gradient methods. A comparison between these iterative schemes is drawn. The numerical results show that the finite difference method is more efficient than the finite element method for regular domains, while the finite element method is more accurate for complex and irregular domains. Moreover, we observe that the Conjugate Gradient iterative technique gives the most efficient results among the other iterative methods.

## Introduction

Partial differential equations (PDEs) are encountered in physics either elliptic, parabolic or hyperbolic. Hyperbolic partial differential equations (PDEs) play a very important role in science, technology and arise very frequently in physical applications as models of waves, such as acoustic, elastic, seismic, shock, electromagnetic, and gravitational waves.

In fact, most of hyperbolic partial differential equations (PDEs) that arise in mathematical models of physical phenomena are very difficult to solve analytically, so numerical methods become necessary to approximate the solution of such hyperbolic partial differential equations. For many hyperbolic partial differential problems, finite difference and finite element methods are the techniques of choice [19].

Finite difference method (FDM) is the oldest method for numerical solution of partial differential equations which is introduced by Euler in the 18th century. Because of this simplicity and easy to use for simple geometries, it is the most popular method for solving partial differential equations. It is based upon the application of Taylor expansion to approximate the differential equations. This method uses a topologically square network of lines to construct the discretization of the PDE [38].

On the other hand, the finite element method (FEM) is the general method for the numerical solution of partial differential equations covering all three main types of equations, namely elliptic, parabolic, and hyperbolic equations. It can be implemented to any type of PDE. FEM is flexible and

accurate method but requires a good knowledge in coding. The finite element method (FEM) is good enough to give a vision of numerical solution [40], and it was introduced by engineers in the late 50's and early 60's for the numerical solution of partial differential equations in structural engineering (elasticity equations, plate equations ).

Today, the method is used extensively for problems in many areas including, but not limited to, structural engineering, strength of materials, fluid mechanics, nuclear engineering, electro-magnetism, convention-diffusion processes, wave propagation, scattering, integrated circuits, heat conduction, petroleum engineering, and reaction-diffusion processes [39].

The FEM dates back to 1909 when Ritz developed an effective method for the approximate solution of problems in the mechanics of deformable solids, it includes an approximation of energy function by the known functions with unknown coefficients. Minimization of function in relation to each unknown leads to a system of equations from which the unknown coefficients may be determined. One of the main restrictions in the Ritz method is that functions used should satisfy the boundary conditions of the problem [8].

The FEM obtained its real impetus in the 1960s and 1970s by the developments of the following groups: J. H. Argyris with co-workers at the University of Stuttgart, R.W. Clough with co-workers at UC Berkeley, O. C. Zienkiewicz with co-workers Ernest Hinton, Bruce Irons and others at the University of Swansea, Philippe G. Ciarlet at the University of Paris and Richard Gallagher with co-workers at Cornell

University[8]. The FEM for solving the wave equation has been developed by many researchers. Very recently, Bangerth and Rannacher [2] have used the finite element approximation for the acoustic wave equation. Hui [12] has implemented the FEM of the elastic wave equation and wave fields simulation in two-phase anisotropic media. Margrave and Mahmodian [22] have applied the FEM in seismic wave modeling. Glowinski and Lapin [10] have obtained the solution of a wave equation by a mixed finite element - fictitious domain method. Ham and Bathe [11] have used the finite element method for wave propagation problems.

On the other hand, the FDM was invented by a Chinese scientist named Feng Kang in the late 1950's. He proposed the FDM as a systematic numerical method for solving partial differential equations that are applied to the computations of dam constructions. It is now considered that the invention of the FDM is a milestone of computational mathematics [38].

The FDM for solving the wave equation has been developed by many researchers. Very recently, Oliveira [26] has used the fourth-order FDM for the acoustic wave equation on irregular grids. Maupin and Dmowska [23] have implemented the finite-difference time-domain method for modeling of seismic wave propagation. Lamoureux et al. [17] have used the Galerkin methods for numerical solutions of acoustic, elastic and viscoelastic wave equations. Chua and Stoffab [7] have studied the Nonuniform grid implicit spatial finite difference method for acoustic wave equation. Lines et al. [20] have analyzed the stability of finite difference wave equations computations. Saarelma [32] has used the finite difference

time domain solver for room acoustics using graphics processing units. Antunes et al. [1] have applied the FDM to solve acoustic wave equation using locally adjustable time-steps. Moczo et al. [24] have investigated the accuracy of the finite difference and the finite element schemes with respect to p-wave to s-wave speed ratio. Dong et al. [9] have applied the finite element and finite difference methods to solve 2D wave equation.

The thesis is organized as follows: Chapter one introduces the basics of the FDM and the FEM for homogeneous and inhomogeneous wave equations with different types of boundary conditions. Chapter two presents some iterative methods namely: Jacobi, Gauss-Seidel, Successive over Relaxation (SOR), and Conjugate Gradient method for solving linear system which is implemented by using the FDM and the FEM and their convergence properties. Chapter three contains some numerical examples and results and finally the conclusions follows.

## Chapter One

### Finite Difference and Finite Element Methods for Solving Hyperbolic Partial Differential Equations

A second order linear partial differential equation is mainly considered as

$$A u_{xx} + B u_{xy} + C u_{yy} + D u_x + E u_y + F u = G(x, y) \quad (1.1)$$

where  $A, B, C, D, E, F$  and the free term  $G$  can either be constants or functions of the two independent variables  $x$  and  $y$ .

Equation (1.1) is classified into three types depending on the discriminant ( $B^2 - 4AC$ ) as follows (see [6]):

1. Hyperbolic if the discriminant is positive ( $B^2 - 4AC > 0$ ).
2. Parabolic if the discriminant is zero ( $B^2 - 4AC = 0$ ).
3. Elliptic if the discriminant is negative ( $B^2 - 4AC < 0$ ).

In this work, we will deal with hyperbolic PDEs (see [19],[37] ).

We will use the Finite Difference and the Finite Element methods for solving hyperbolic partial differential equation for both homogeneous and inhomogeneous wave equations.

To implement these methods to solve the hyperbolic PDE, a system of linear equations will be generated that can be solved using several iterative schemes such as Jacobi, Gauss-Seidel, Successive over Relaxation (SOR), and Conjugate Gradient methods.

### 1.1 Hyperbolic PDE Subject to Boundary Conditions

Solution of homogeneous and inhomogeneous wave equation on the boundary of a domain  $D$  needs certain conditions where the unknown function (dependent variable) must satisfy these conditions on the boundary  $B$ . We will deal with homogeneous and inhomogeneous wave equations with respect to two types of boundary conditions . These boundary conditions are:

#### 1. Dirichlet Boundary Conditions:

The condition where the value of the unknown function is prescribed on the boundary of the domain.

#### 2. Neumann Boundary Conditions:

The condition where the value of the normal derivative  $\frac{\partial u}{\partial n}$  is given on the boundary of the domain.

### 1.2 Discretization of Hyperbolic PDE by Finite Difference Method

This method is effective when the domain of the problem has boundaries with regular shapes. In this thesis, we will deal with the FDM with rectangular domain of regular boundaries shapes (see [35]).

### 1.3 The Principle of Finite Difference Method (FDM)

The FDM is a numerical method for solving differential equations by approximating them using difference equations with errors of order  $h^n$  ( $O(h^n)$ ).



The given region or domain of the PDE is divided into a network of lines constructing rectangles called grid. The points of intersection of these lines are called grid points or mesh points. At each grid point, the differential equation is approximated by replacing the partial derivatives by their corresponding difference approximations. The replacement of partial derivatives with difference approximation formulas depends on Taylor's Theorem . This gives an algebraic equation for each grid point, in which the variable value at that point and a certain number of neighbour points appears as unknown. In other words, by knowing the value of the variable at neighbouring points of the unknown value makes that variable at that particular point can be calculated (see [19], [28] ).

#### **1.4 Strategy of Discretization**

Using the FDM to discretize hyperbolic PDE with its boundary conditions, we can consider the following inhomogeneous wave equation:

$$c^2 u_{xx} - u_{tt} = G(x, t) \quad (1.2)$$

The rectangular domain  $D = \{(x, t) \mid a < x < b, c < t < d\}$  and

$u(x, t) = g(x, t)$  for any  $(x, t) \in B$ , where  $B$  denotes the boundary of a region  $D$ ,  $G(x, t)$  is a continuous function on  $D$  and  $g(x, t)$  is continuous on  $B$ . The continuity of both  $G$  and  $g$  guarantees a unique solution of equation (1.2) (see [13],[ 19]).

Now, we will use the finite difference algorithm for solving hyperbolic PDE, like equation (1.2).

### The Finite Difference Algorithm

**Step 1:** Choose positive integers  $n$  and  $m$ .

**Step 2:** Define  $h = \frac{b-a}{n}$  and  $k = \frac{d-c}{m}$

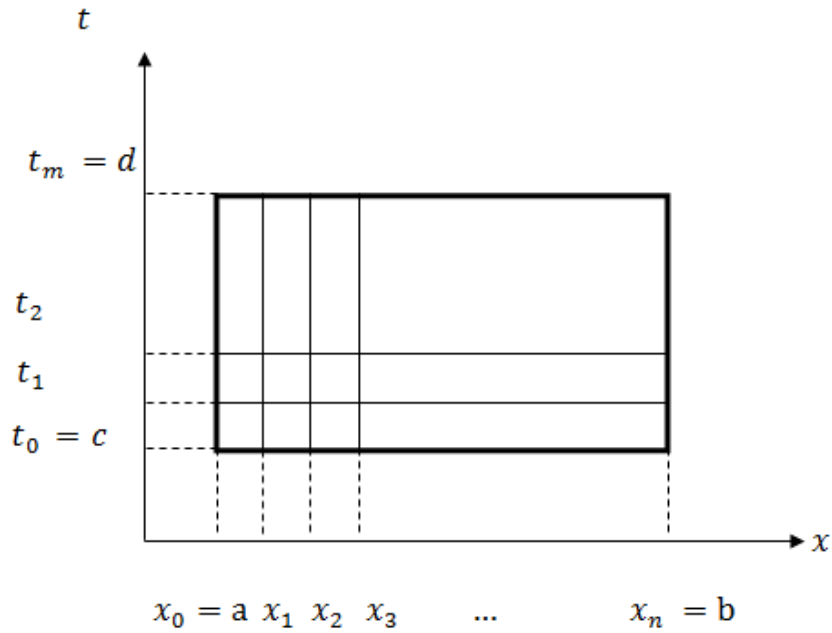
This step partitions the interval  $[a, b]$  into  $n$  equal parts of width  $h$  and partitions the interval  $[c, d]$  into  $m$  equal parts of width  $k$ .

**Step 3:** Define the mesh point  $(x_i, t_j)$  as

$$x_i = a + ih, i = 1, 2, \dots, n$$

$$t_j = c + jk, j = 1, 2, \dots, m$$

Step 2 and step 3 are illustrated in figure 1.1.



**Figure1. 1:** discretization of a rectangular domain

At any interior mesh point  $(x_i, t_j)$  the wave equation becomes

$$c^2 u_{xx}(x_i, t_j) - u_{tt}(x_i, t_j) = G(x_i, t_j) \quad (1.3)$$

The difference method is obtained using the centered-difference quotient for the second partial derivatives given by (see [29]):

$$u_{xx}(x_i, t_j) = \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j))}{h^2} - \frac{h^2}{12} \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j), \text{ where } \xi_i \in (x_{i-1}, x_{i+1}) \quad (1.4)$$

and

$$u_{tt}(x_i, t_j) = \frac{u(x_i, t_{j+1}) - 2u(x_i, t_j) + u(x_i, t_{j-1}))}{k^2} - \frac{k^2}{12} \frac{\partial^4 u}{\partial t^4}(x_i, \mu_j), \text{ where } \mu_j \in (t_{j-1}, t_{j+1}) \quad (1.5)$$

Substituting equations (1.4) and (1.5) into equation (1.3) gives

$$c^2 \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j))}{h^2} - \frac{u(x_i, t_{j+1}) - 2u(x_i, t_j) + u(x_i, t_{j-1}))}{k^2} - \frac{1}{12} \left[ c^2 h^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) - k^2 \frac{\partial^4 u}{\partial t^4}(x_i, \mu_j) \right] = G(x_i, t_j) \quad (1.6)$$

for each  $i = 1, 2, 3, \dots, n-1$  and  $j = 1, 2, 3, \dots, m-1$ .

The boundary conditions are:

1.  $u(x_0, t_j) = g(x_0, t_j)$ , for  $j = 0, 1, 2, \dots, m$ .
2.  $u(x_n, t_j) = g(x_n, t_j)$ , for  $j = 0, 1, 2, \dots, m$ . (1.7)

$$3. (x_i, t_0) = g(x_i, t_0), \text{ for } i = 1, 2, \dots, n-1.$$

$$4. (x_i, t_m) = g(x_i, t_m), \text{ for } i = 1, 2, \dots, n-1.$$

Now, by rearranging equation (1.6), we get:

$$\begin{aligned} & c^2 \frac{u(x_{i+1}, t_j) + u(x_{i-1}, t_j)}{h^2} - \frac{u(x_i, t_{j+1}) + u(x_i, t_{j-1})}{k^2} - 2 \frac{c^2}{h^2} u(x_i, t_j) \\ & + 2 \frac{1}{k^2} u(x_i, t_j) = \frac{1}{12} \left[ c^2 h^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) - k^2 \frac{\partial^4 u}{\partial t^4}(x_i, \mu_j) \right] + G(x_i, t_j) \end{aligned}$$

Or it can simply be written as

$$\begin{aligned} & c^2 \frac{u(x_{i+1}, t_j) + u(x_{i-1}, t_j)}{h^2} - \frac{u(x_i, t_{j+1}) + u(x_i, t_{j-1})}{k^2} \\ & - 2 \left[ \frac{c^2}{h^2} - \frac{1}{k^2} \right] u(x_i, t_j) \\ & = \frac{1}{12} \left[ c^2 h^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) - k^2 \frac{\partial^4 u}{\partial t^4}(x_i, \mu_j) \right] + G(x_i, t_j) \quad (1.8) \end{aligned}$$

Multiplying both sides by  $k^2$ , we get:

$$\begin{aligned} & \left( \frac{ck}{h} \right)^2 [u(x_{i+1}, t_j) + u(x_{i-1}, t_j)] - u(x_i, t_{j+1}) - u(x_i, t_{j-1}) \\ & - 2 \left[ \left( \frac{ck}{h} \right)^2 - 1 \right] u(x_i, t_j) \\ & = \frac{k^2}{12} \left[ c^2 h^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) - k^2 \frac{\partial^4 u}{\partial t^4}(x_i, \mu_j) \right] + k^2 G(x_i, t_j) \quad (1.9) \end{aligned}$$

Define  $\lambda = ck/h$  and neglecting the error term (local truncation error) defined as :

$O(h^2 + k^2)$ :  $\tau_{i,j} = c^2 h^2 \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) - k^2 \frac{\partial^4 u}{\partial t^4}(x_i, \mu_j)$ . Then Simplifying equation (1.9) and letting  $u_{i,j}$  approximate  $u(x_i, t_j)$  we can write the difference equation as

$$\lambda^2 (u_{i+1,j} + u_{i-1,j}) - u_{i,j+1} - u_{i,j-1} + 2(1 - \lambda^2) u_{i,j} = k^2 G(x_i, t_j)$$

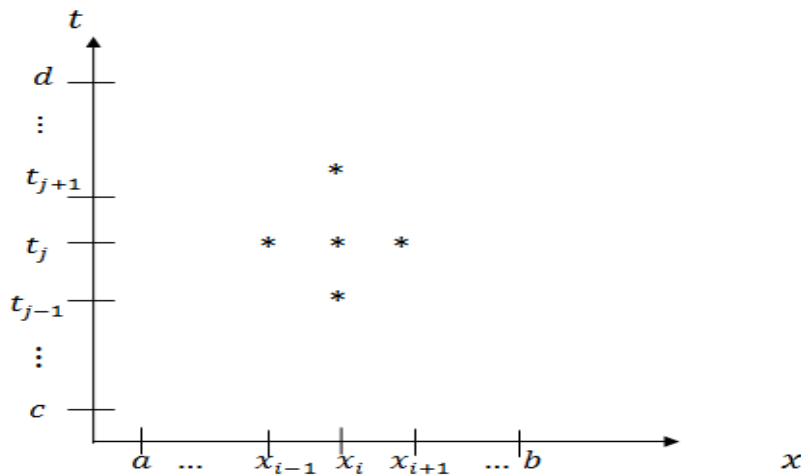
for each  $i = 1, 2, \dots, n-1$  and  $j = 1, 2, \dots, m-1$ . (1.10)

with boundary conditions:

1.  $u_{0,j} = g(x_0, t_j)$ , for  $j = 0, 1, 2, \dots, m$ .
2.  $u_{n,j} = g(x_n, t_j)$ , for  $j = 0, 1, 2, \dots, m$ . (1.11)
3.  $u_{i,0} = g(x_i, t_0)$ , for  $i = 1, 2, \dots, n-1$ .
4.  $u_{i,m} = g(x_i, t_m)$ , for  $i = 1, 2, \dots, n-1$ .

Equation (1.10) involves approximations to the unknown function  $u(x, t)$  at the points  $(x_{i-1}, t_j), (x_{i+1}, t_j), (x_i, t_{j-1}), (x_i, t_{j+1}), (x_i, t_j)$

These points form a regular star-shape region in the grid (as shown in figure 1.2).



**Figure1. 2:** 5-points stencil of the unknown function  $u(x, t)$

When we use formula (1.10) with boundary conditions (1.11), then at all points  $(x_i, t_j)$  that are adjacent to a boundary mesh point, we have an  $(n - 1) \times (m - 1)$  by  $(n - 1) \times (m - 1)$  linear system with the unknowns being the approximations  $u_{i,j}$  to  $u(x_i, t_j)$  at the interior mesh points.

The generated linear system can be solved by the Jacobi, Gauss-Seidel, Successive over Relaxation (SOR), or Conjugate Gradient methods. This system (that involves the unknowns) produces satisfactory results if a relabeling of the interior mesh points is introduced (see [5],[ 39]). A favorable labeling of these points is:

$$L_r = (x_i, t_j) \text{ and } u_r = u_{i,j}, \text{ where } r = i + (m - 1 - j)(n - 1)$$

$$\forall i = 1, 2, \dots, n - 1, \text{ and } \forall j = 1, 2, \dots, m - 1.$$

#### **1.4.1 Homogeneous Wave Equation with Dirichlet Boundary Conditions**

When the function is defined on any part of a domain  $D$ , then we call this part Dirichlet boundary  $S_D$ , i.e. the unknown function  $u$  is prescribed on the boundary, that is,  $u(x, t) = g(x, t), (x, t) \in B$  where the function  $g$  is a known function .

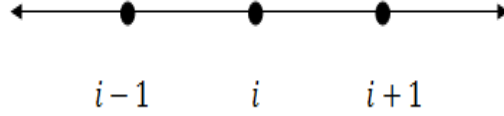
$$B_D: u = g$$

To derive the formula of finite difference approximation with Dirichlet boundary condition for homogeneous wave equation

$$c^2 \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = 0 \tag{1.12}$$

we consider three points  $i + 1, i$ , and  $i - 1$  which are located on  $x$ -axis with equal distance  $h$  between them as shown in figure 1.3 ,

( see [6], [13] and[37]).



**Figure1. 3:** three points  $i + 1, i$ , and  $i - 1$  which are located on  $x$ -axis

The value of the function  $u(x, t)$  at the points  $(i - 1, j), (i, j)$ , and  $(i + 1, j)$  be  $u_{i-1,j}$ ,  $u_{i,j}$ , and  $u_{i+1,j}$ , respectively.

Now, use Taylor series to express  $u_{i-1,j}$  and  $u_{i+1,j}$  in the form of Taylor expansions about the point  $i$  as follows:

$$u_{i+1,j} = u_{i,j} + \frac{h}{1!} \cdot \frac{\partial u}{\partial x} \Big|_i + \frac{h^2}{2!} \cdot \frac{\partial^2 u}{\partial x^2} \Big|_i + \frac{h^3}{3!} \cdot \frac{\partial^3 u}{\partial x^3} \Big|_i + \frac{h^4}{4!} \cdot \frac{\partial^4 u}{\partial x^4} \Big|_i + O(h^5) \quad (1.13)$$

$$u_{i-1,j} = u_{i,j} - \frac{h}{1!} \cdot \frac{\partial u}{\partial x} \Big|_i + \frac{h^2}{2!} \cdot \frac{\partial^2 u}{\partial x^2} \Big|_i - \frac{h^3}{3!} \cdot \frac{\partial^3 u}{\partial x^3} \Big|_i + \frac{h^4}{4!} \cdot \frac{\partial^4 u}{\partial x^4} \Big|_i + O(h^5) \quad (1.14)$$

Adding equations (1.13) and (1.14), gives:

$$u_{i+1,j} + u_{i-1,j} = 2u_{i,j} + h^2 \cdot \frac{\partial^2 u}{\partial x^2} \Big|_i + \frac{h^4}{12} \cdot \frac{\partial^4 u}{\partial x^4} \Big|_i + O(h^5)$$

By rearranging the above equation, we obtain:

$$\frac{\partial^2 u}{\partial x^2} \Big|_i = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2) \quad (1.15)$$

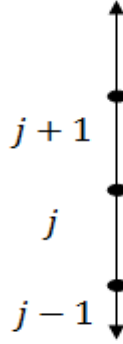
Equation (1.15) is a finite difference approximation formula with the error term  $O(h^2)$ .

Subtracting equation (1.14) from equation (1.13), we get:

$$\frac{\partial u}{\partial x} \Big|_i = \frac{u_{i+1,j} - u_{i-1,j}}{2h} + O(h^2) \quad (1.16)$$

Equation (1.16) is a finite difference approximation formula with the error term  $O(h^2)$ .

Similarly, consider three points  $j + 1, j$ , and  $j - 1$  which are located on the  $y$ -axis with equal distance  $h$  between them. Let the value of the function  $u(x, t)$  at the points  $(i, j - 1)$ ,  $(i, j)$ , and  $(i, j + 1)$  be  $u_{i,j-1}$ ,  $u_{i,j}$ , and  $u_{i,j+1}$ , respectively as shown in figure 1.4.



**Figure 1. 4:** three points  $j + 1, j$ , and  $j - 1$  which are located on the  $y$ -axis

Using Taylor series to express  $u_{i,j+1}$  and  $u_{i,j-1}$  in the form of Taylor expansions about the point  $j$ , then the finite difference approximation formulas with the error term  $O(h^2)$  of second order for  $\frac{\partial^2 u}{\partial x^2} \Big|_j$  and  $\frac{\partial u}{\partial x} \Big|_j$  are, respectively:



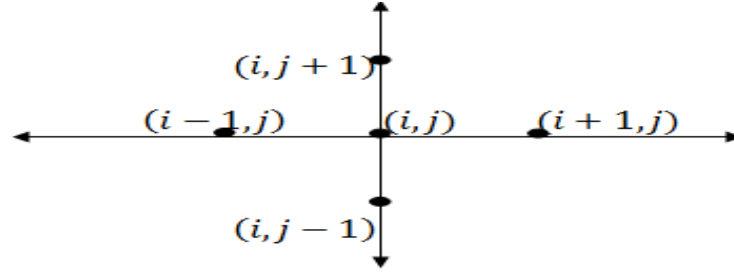
$$\frac{\partial^2 u}{\partial t^2} \Big|_j = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} + O(h^2) \quad (1.17)$$

and

$$\frac{\partial u}{\partial t} \Big|_j = \frac{u_{i,j+1} - u_{i,j-1}}{2h} + O(h^2) \quad (1.18)$$

Combining  $x$  and  $t$  axis together, we get the star-shape

( 5-points stencil) region about the point  $(i, j)$  as shown in figure 1.5 .



**Figure1. 5:** combining x-axis and t-axis around the  $(i, j)$  point

Inserting equations (1.15) and (1.17) into equation (1.12) yields:

$$\left( c^2 \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} \right) \Big|_{(i,j)} = c^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} - \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0$$

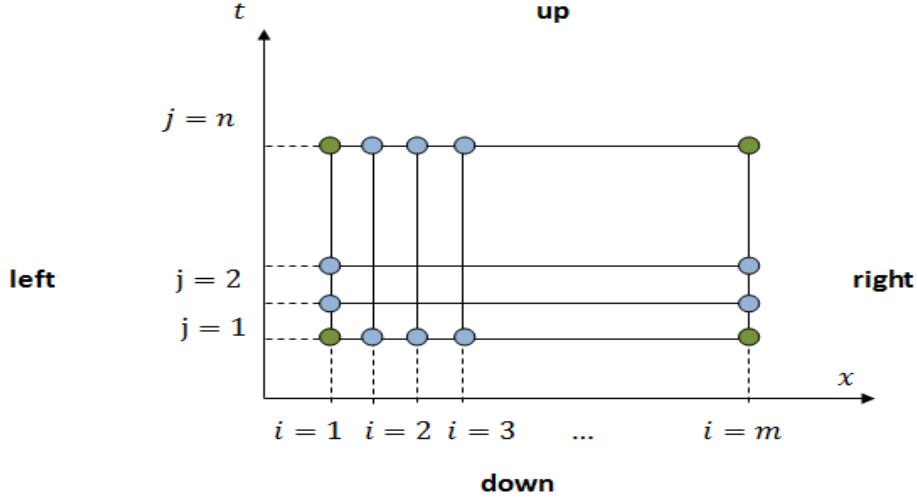
Rearranging the above equation, we get

$$c^2 u_{i+1,j} + c^2 u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 2(1 - c^2) u_{i,j} = 0 \quad (1.19)$$

$$u_{i,j} = \frac{-1}{2(c^2-1)} [u_{i,j+1} + u_{i,j-1} - c^2 u_{i+1,j} - c^2 u_{i-1,j}] \quad (1.20)$$

In general, if  $u$  satisfies the wave equation, then  $u$  at any point in the domain  $D$  satisfies equation (1.20).

Now, suppose we have Dirichlet boundary conditions defined on the rectangular domain such that  $1 \leq i \leq m$  and  $1 \leq j \leq n$  (see [6]) as shown in figure 1.6 .



**Figure 1. 6:** Dirichlet boundary conditions defined on the rectangular domain

Let  $u(x, t) = g(x, t)$  be given on all boundaries of the domain, that is  $u = g$  is defined on the left, up, right and down boundary walls so that the boundary grid points (blue points) and the corner grid points (green points) are known. In other words, the values of the points  $(x_i, t_j)$ ,  $\forall i = 2, 3, \dots, m-1$ ,  $\forall j = 2, 3, \dots, n-1$  ( see [6],[34]) under the function  $g$  are known. For the corner grid points, we use the following equations

$$\begin{aligned}
 u(1,1) &= \frac{1}{2}[u(2,1) + u(1,2)] \\
 u(m,1) &= \frac{1}{2}[u(m-1,1) + u(m,2)] \\
 u(1,n) &= \frac{1}{2}[u(1,n-1) + u(2,n)] \\
 u(m,n) &= \frac{1}{2}[u(m,n-1) + u(m-1,n)]. \quad (1.21)
 \end{aligned}$$

### 1.4.2 Inhomogeneous Wave Equation with Dirichlet Boundary Conditions

To derive the formula of finite difference approximation with Dirichlet boundary conditions for inhomogeneous wave equation:

$$c^2 \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = G(x, t) \quad (1.22)$$

We follow similar approach for the homogeneous wave equation with some amendments in equation (1.19), that is

$$u_{i,j} = \frac{-1}{2(c^2 - 1)} [u_{i,j+1} + u_{i,j-1} - c^2 u_{i+1,j} - c^2 u_{i-1,j} - h^2 G_{i,j}] \quad (1.23)$$

### 1.4.3 Homogeneous Wave Equation with Neumann Boundary Conditions

When the normal derivative of the unknown function  $u$  is prescribed on the boundary of a domain  $D$ , then we call this part Neumann boundary  $B_N$ , i.e. the value of the normal derivative of the function is given on the boundary of the domain, where  $g(x, t)$  is a given function.

$$B_N: \frac{\partial u}{\partial n} = g(x, t)$$

To derive the formula of finite difference approximation with Neumann boundary condition for the wave equation

$$c^2 \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = 0$$

Consider that we have a rectangular domain as shown in figure 1.6.

Suppose that Dirichlet condition is specified on up, right and down walls and Neumann condition is defined on the remaining wall which is the left wall as follows (see [27] ):

$$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial x} = -g(t) \quad (1.24)$$

Now, we want to approximate equation (1.24) using the second order approximation using equation (1.16). This procedure puts the grid points  $(1, j)$  outside the domain towards the left that is located on imaginary boundary that their fake coordinates will be  $(0, j)$  (see [16],[27]).

So, equation (1.24) is approximated using equation (1.16) at the line  $i = 1$

$$\frac{\partial u}{\partial x} \big|_{(1,j)} = \frac{u_{1+1,j} - u_{1-1,j}}{2h} = \frac{u_{2,j} - u_{0,j}}{2h} = -g(1, j)$$

Thus,

$$u_{0,j} = u_{2,j} + 2h g(1, j) \quad (1.25)$$

Now, we write equation (1.20) at the point  $(1, j)$  as

$$u_{1,j} = \frac{1}{2(c^2 - 1)} [u_{1,j+1} + u_{1,j-1} - c^2 u_{1+1,j} - c^2 u_{1-1,j}]$$

$$u_{1,j} = \frac{1}{2(c^2 - 1)} [u_{1,j+1} + u_{1,j-1} - c^2 u_{2,j} - c^2 u_{0,j}] \quad (1.26)$$

Substituting equation (1.25) into equation (1.26), we get:

$$\begin{aligned}
u_{1,j} &= \frac{1}{2(c^2 - 1)} [u_{1,j+1} + u_{1,j-1} - c^2 u_{2,j} - c^2 u_{2,j} - 2c^2 h g(1,j)] \\
u_{1,j} &= \frac{1}{2(c^2 - 1)} [u_{1,j+1} + u_{1,j-1} - 2c^2 u_{2,j} - 2c^2 h g(1,j)] \quad (1.27)
\end{aligned}$$

For any two positive integers  $m$  and  $n$ , we use equation (1.27) for

$2 \leq j \leq n - 1$ , where  $g(1,j)$  is a specified function. As Dirichlet condition is specified on up, right, and down walls, the values

$\{u(i, n), 2 \leq i \leq m - 1\}, \{u(m, j), 2 \leq j \leq n - 1\}$  and

$\{u(i, 1), 2 \leq i \leq m - 1\}$  are known. To find the values of corner grid points, we use equation (1.21).

#### 1.4.4 Inhomogeneous Wave Equation with Neumann Boundary Conditions

Consider the inhomogeneous wave equation:

$$c^2 \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = G(x, t)$$

with Neumann boundary condition:

$$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial t} = -g(t)$$

defined on the rectangular domain.

Similar to the homogeneous wave equation, the difference approximation formula of Neumann condition at the fake grid point  $(0, j)$  is equation (1.25), that is (see [27]):

$$u_{0,j} = u_{2,j} + 2h g(1, j)$$

Now, using equation (1.24) to find the value of the point  $(1, j)$ , we get

$$u_{1,j} = \frac{1}{2(c^2-1)} [u_{1,j+1} + u_{1,j-1} - c^2 u_{2,j} - c^2 u_{0,j} - h^2 G_{1,j}] \quad (1.28)$$

Substituting equation (1.25) into equation (1.28), we get

$$u_{1,j} = \frac{1}{2(c^2-1)} \left[ u_{1,j+1} + u_{1,j-1} - c^2 u_{2,j} - c^2 (u_{2,j} + 2h g(1,j)) - h^2 G_{1,j} \right]$$

Thus

$$u_{1,j} = \frac{1}{2(c^2-1)} [u_{1,j+1} + u_{1,j-1} - 2c^2 u_{2,j} - 2hc^2 g(1,j) - h^2 G_{1,j}] \quad (1.29)$$

If  $i \neq 1$ , we use equation (1.23).

Using the same method, we can deal with other boundary points except the corner points. For corner points, we use equation (1.21) to find their values (see [35],[36] ).

### 1.5 Finite Element Method

The Finite Element Method (FEM) is the most known numerical method used for solving partial differential equations to approximate the solution of them when the analytical (exact) solution is impossible to find. This method is effective when the domain of the problem has boundary with irregular shapes. Finite element method (FEM) can be applied on many scientific and engineering problems such as fluid flow, heat transfer, electromagnetic fields, aerospace, civil engineering, and so on (see [30] ).

### **The Principle of Finite Element Method (FEM)**

The following are the basic steps involved in the finite element method ( see [15] and [33] ):

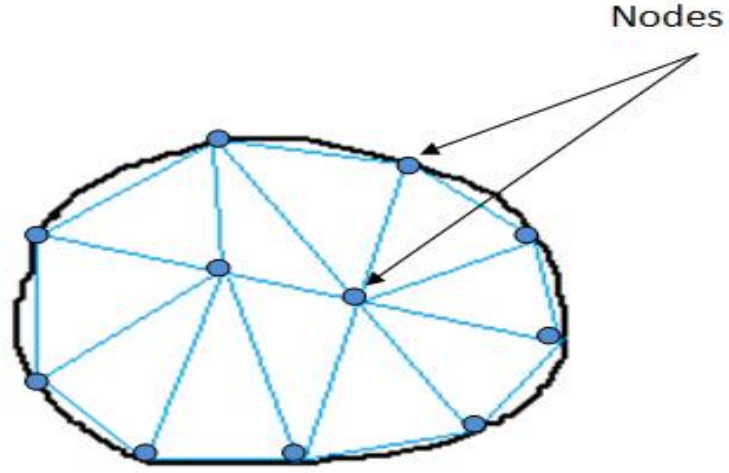
**Discretization:** The discretization of the given differential equation is obtained by dividing the given domain  $D$  into a finite number of elements . The points at which those finite elements intersect are called nodes (blue points as in figure 1.7). The nodes and elements both are numbered by a suitable indices .

**Derivation of finite element equations:** For any given differential equation, a variational formulation is constructed for each element. Then the element equations are obtained by substituting a typical dependent variable into the variational formulation. After choosing the variable  $i$  and the interpolation functions, the element matrices can be computed.

**Assembly:** After the calculation of element matrices, the next step is to assemble those element equations so that the final solution is continuous. When this assembly is done, the entire system of equations takes the matrix form .

**Boundary conditions:** Apply the boundary conditions for that problem to the above system of equations .

**Solution of the equations:** Finally the system is solved by any available standard technique for solving system of equations, for example Gauss elimination.



**Figure1. 7:** finite element on irregular shape

### 1.5.1 Finite Element Method (FEM) for Dirichlet Boundary Value Problems

This section discusses the finite element method that is used to solve one dimensional hyperbolic partial differential equation with Dirichlet boundary conditions in a rectangular domain and focuses on finite element solution using spreadsheets with triangular grid (see [15],[33]).

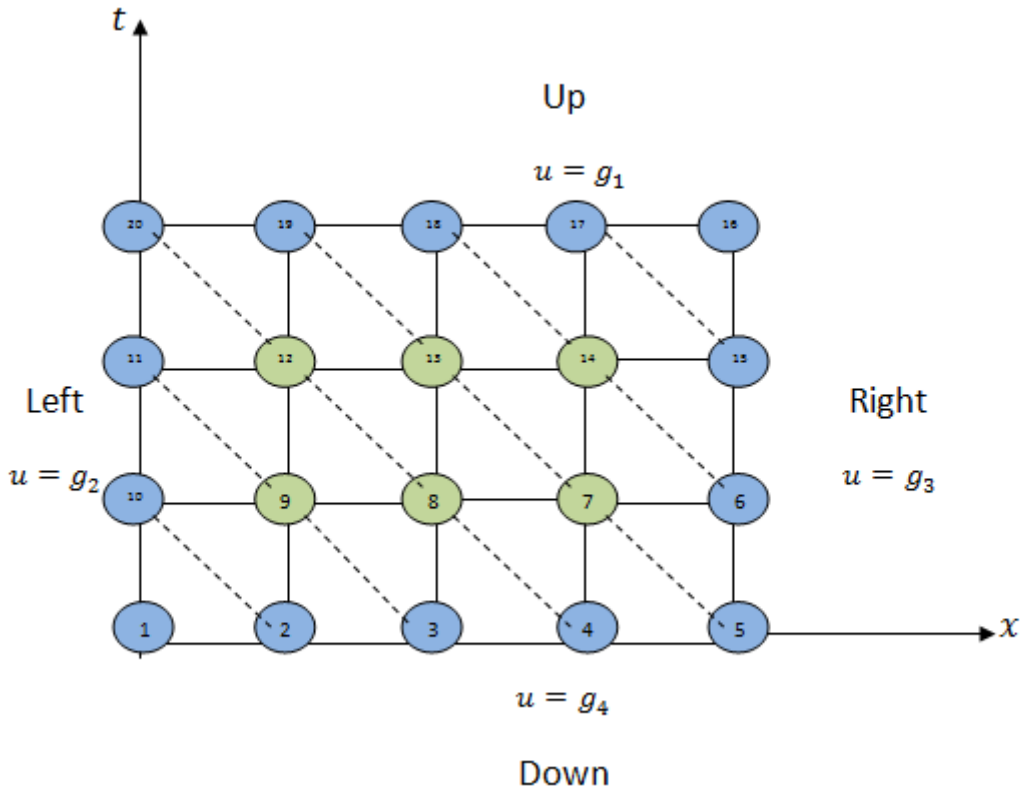
Now, we want to approximate the solution of homogeneous wave equation

$$c^2 u_{xx} - u_{tt} = 0$$



defined on a rectangular domain with Dirichlet boundary conditions defined on the up, left, right and down boundaries (edges) as shown in figure 1.8.

The region is divided into equal triangular elements. In this discretization, there are global nodes such that the nodes which are located on the boundaries (blue nodes) that the function  $u$  defined on them is known and interior nodes (green nodes) that the function  $u$  defined on them is unknown.



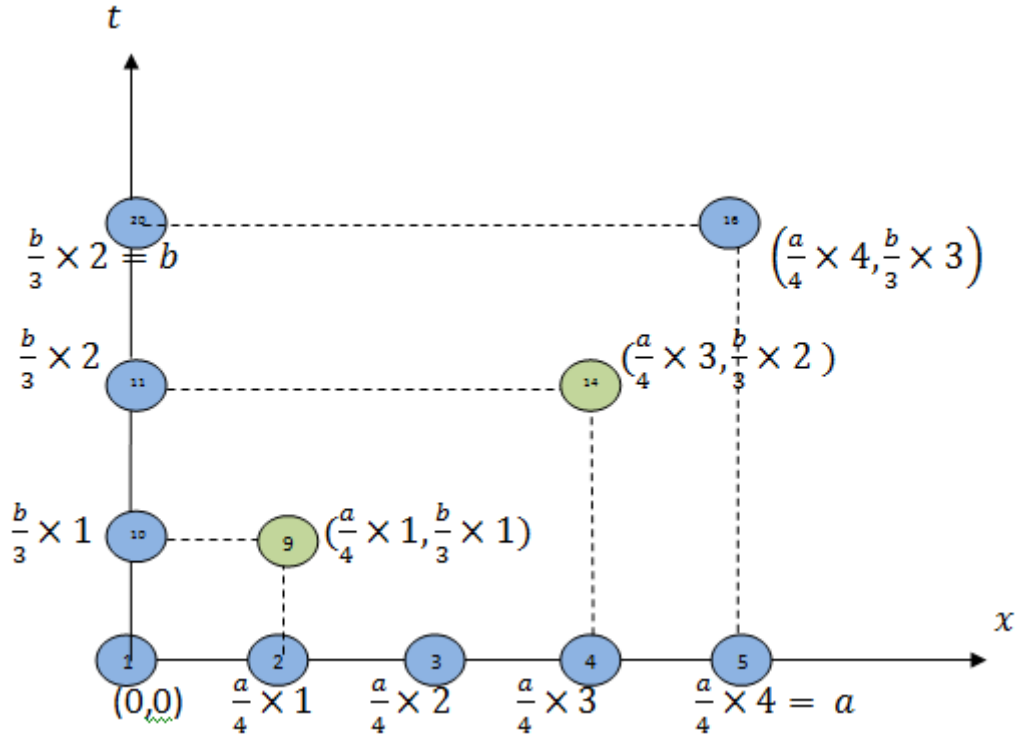
**Figure1. 8:** rectangular domain with Dirichlet boundary conditions

Suppose  $0 < x < a$  and let  $m$  be the number of equal portions that are located on bottom boundary ( $x$ -axis). In this case,  $m = 4$  portions (from node 1 to node 2, from node 2 to node 3, from node 3 to node 4

and from node 4 to node 5). The length of each portion is equal to the length of other portions and is equal to  $\frac{a}{m} = \frac{a}{4}$ .

In similar manner, suppose  $0 < t < b$  and let  $n$  be the number of equal portions that are located on the left boundary ( $t$ -axis).

In this case,  $n = 3$  portions (from node 1 to node 10, from node 10 to node 11 and from node 11 to node 20). The length of each portion is equal to the length of other portions and is equal to  $\frac{b}{n} = \frac{b}{3}$ , (see [30]).



**Figure1. 9:** coordinate for each node in finite element method

Now, we can easily find the coordinate for each node as shown in figure 1.9 as follows:

Node 1: (0,0)

Node 9:  $(\frac{a}{4} \times 1, \frac{b}{3} \times 1)$

Node 14:  $(\frac{a}{4} \times 3, \frac{b}{3} \times 2)$  and so on until Node 18 :  $(a, b)$ .

Now, for each element (triangle)  $e$ , we determine the local node numbers 1, 2, and 3 that must be assigned so that global nodes associated with an element are traversed in a counterclockwise sense.

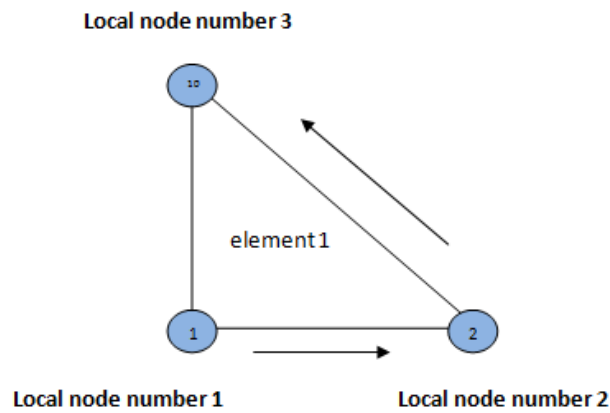
For example: Element 1:

At node 1: the local node number is 1, so  $(x_1, t_1) = (0, 0)$

At node 2: the local node number is 2, so  $(x_2, t_2) = (\frac{a}{4}, 0)$

At node 10: the local node number is 3, so  $(x_3, t_3) = (0, \frac{b}{3})$

These are shown in figure 1.10.



**Figure1. 10:** the local node numbers are determined on nodes start from node1, then node 2 and finally with node 10 (in a counterclockwise).

Similarly, we determine the local node numbers 1, 2 and 3 for each element  $e_i$  in the same way as in element 1.

The following must be computed for each element  $e_i$  :

$$P_1 = t_2 - t_3 \quad Q_1 = x_3 - x_2$$

$$P_2 = t_3 - t_1 \quad Q_2 = x_1 - x_3$$

$$P_3 = t_1 - t_2 \quad Q_3 = x_2 - x_1$$

For element 1:

$$(x_1, t_1) = (0, 0), (x_2, t_2) = \left(\frac{a}{4}, 0\right), (x_3, t_3) = \left(0, \frac{b}{3}\right)$$

$$P_1 = t_2 - t_3 = 0 - \frac{b}{3} = -\frac{b}{3}$$

$$Q_1 = x_3 - x_2 = 0 - \frac{a}{4} = -\frac{a}{4}$$

and so on.

Now, for each element  $e$ , we want to find the  $3 \times 3$  element coefficient matrix for which the entries are given by the equation:

$$C_{ij}^{(e)} = \frac{1}{4A} [P_i P_j + Q_i Q_j], \text{ for } i, j = 1, 2, 3, \quad (1.30)$$

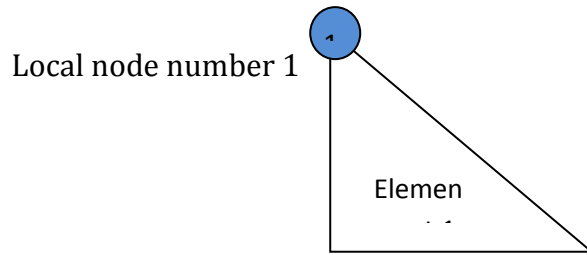
where

$$A = \frac{1}{2} [P_2 Q_3 - P_3 Q_2].$$

When we find the element coefficient matrices, then the global coefficient matrix  $C$  is assembled from the element coefficient matrices. If the number of nodes is  $N$ , then the global coefficient matrix  $C$  will be an  $N \times N$  matrix (in our case,  $N=20$ ).

We can compute the entries of the main diagonal as follows:

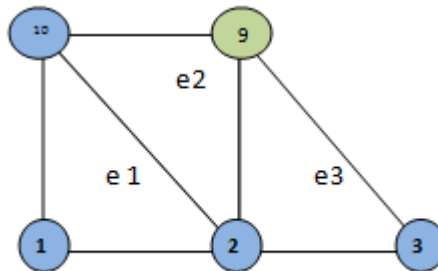
$C_{1,1}$ : is the entry that is located on row 1 and column 1 in the global coefficient matrix  $C$  which corresponds to node 1 that belongs to element 1 only. Node 1 is assigned local node number 1 in element 1 as shown in the following figure (1.11).



**Figure1. 11:** Node 1 is assigned local node number 1 in element 1

$C_{1,1} = C_{11}^{(1)}$  is the entry that is located on row 1 and column 1 in the element coefficient matrix for element 1.

$C_{2,2}$ : is the entry that is located on row 2 and column 2 in the global coefficient matrix  $C$  which corresponds to node 2 that belongs to elements 1, 2, and 3. Node 2 is assigned local node number 2 in element 1 and local node number 1 in elements 2 and 3 as shown in the following figure(1.12), (see [33] ).



**Figure1. 12:** Node 2 has Local node number 2 in element 1 and Local node number 1 in element 2 and element 3.

$C_{2,2} = C_{22}^{(1)} + C_{11}^{(2)} + C_{11}^{(3)}$  where is the entry  $C_{22}^{(1)}$  that is located on row 2 and column 2 in the element coefficient matrix for element 1 and  $C_{11}^{(2)}, C_{11}^{(3)}$  are the entries that are located on row 1 and column 1 in the element coefficient matrix for element 2 and element 3, respectively.

Using the same method, we can find the remaining diagonal entries , for  $i = 1, \dots, N$ . For other entries in the global coefficient matrix  $C$ , we do that using a different method described in the following paragraphs:

Take, for example, the entry  $C_{2,9}$  in the global coefficient matrix  $C$ . It corresponds to node 2 and node 9. So, the link between node 2 and node 9 is called global link which corresponds to local link 1–2 of element 2 and local link 1–3 of element 3 as shown in figure 1.12.

Hence,

$$C_{2,9} = C_{12}^{(2)} + C_{13}^{(3)}$$

The other off-diagonal entries are treated similarly.

Now, defining vector  $u_v$  to be a vector of unknowns (interior nodes, green nodes) and vector  $u_n$  to be a vector of prescribed boundary values. In other words, is a vector of the value of nodes that are located on the boundaries (blue nodes) as shown in figure 1.8.

Define matrix  $C_{vv}$  to be a matrix of unknown nodes obtained from the global coefficient matrix  $C$  and matrix  $C_{vn}$  to be a matrix of unknown

nodes and prescribed boundary values that is also obtained from the global coefficient matrix  $C$ .

In our case,  $C_{vv}$  is a  $6 \times 6$  matrix since we have 6 interior nodes (green nodes) and  $C_{vn}$  is a  $6 \times 14$  matrix since we have 6 interior nodes (green nodes) and 14 boundary nodes (blue nodes) as shown in figure 1.8.

The vector  $u_v$  of unknown nodes can be computed by using:

$$u_v = -C_{vv}^{-1}C_{vn}u_n \quad (1.31)$$

The vector  $u_v$  contains the approximations to the unknown nodes (interior nodes), (see [33] ).

### **1.5.2 Finite Element Method (FEM) with Neumann Boundary condition**

We derive and analyze a finite element method for the 1D wave equation

$$c^2 \Delta u - u_{tt} = f \text{ on } \Omega \quad (1.32)$$

with boundary conditions  $u = 0$  on  $\Gamma$  where  $\Omega$  is a bounded domain in the plane with boundary  $\Gamma$ ,  $f$  is a given real-valued piecewise continuous bounded function in  $\Omega$ .

#### **Finite Element Discretization**

We formulate a finite element method for equation (1.32) based on using continuous piecewise linear function in space (see [3], [4], [40]).

Define the following subspace of a Sobolev space:

$V = \{v(x, t) \mid v \text{ is a continuous function on } \Omega, \alpha x \text{ and } \alpha t \text{ are piecewise continuous and bounded on } \Omega, \text{ and } \alpha = 0 \text{ on } \Gamma\}.$

$$\int_{\Omega} (c^2 \Delta u - u_{tt} - f)v \, dx = 0, \quad \forall v \in V \quad (1.33)$$

The goal of the following development is to simplify equation (1.33) in the general case, and then to switch focus on a finite-dimensional subspace  $V_h$  of  $V$  spanned by the basis functions  $\phi_i(x)$ . This will allow us to write out a linear system of equations for the unknown coefficients  $u(t)$ . It is easy to verify by Gauss' divergence theorem that

$$\int_{\Omega} v \Delta u \, dx = - \int_{\Omega} \nabla v \cdot \nabla u \, dx + \int_{\partial\Omega} v \cdot \nabla u \, d\sigma \quad (1.34)$$

where  $d\sigma$  is an area element on  $\partial\Omega$ . Applying this result to equation (1.33) we have

$$\int_{\Omega} (c^2 \nabla u \cdot \nabla v - u_{tt}v - fv) \, dx - \int_{\partial\Omega} v \nabla u \cdot n \, d\sigma = 0 \quad (1.35)$$

which is called the variational formulation of the wave equation. The solutions to equation (1.35) are called weak solutions to the wave equation (see [18]).

Now, suppose the domain  $\Omega$  is divided into finite number of elements (triangles)  $T_i, \forall i = 1, 2, \dots, m$  such that:

$$\bar{\Omega} = \bigcup_{i=1}^m \bar{T}_i, \text{ where } \bar{\Omega} = \Omega \cup \Gamma.$$

Let  $T_h$  be a partition of  $\Omega$ . Take any triangle  $T \in T_h$  where:



$\text{diam}(T)$  = the longest edge of  $\bar{T}$  and  $h = \max_{T \in T_h} \text{diam}(T)$ .

Now, we can define the finite element space as follows:

$V_h = \{v(x, t) \mid v \text{ is a continuous function on } \Omega, \text{ and it is linear on each triangle } T \in T_h, \text{ and } \alpha = 0 \text{ on } \Gamma\}$  (see [18]).

Each triangle  $T_i, \forall i = 1, 2, \dots, m$  has three vertices denoted by  $v_1, v_2, v_3$ .

We define the basis function  $\phi_i$  as follows:

$$\phi_i(v_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

$$\forall i = 1, 2, \dots, m \text{ and } j = 1, 2, 3.$$

Let  $v$  be a set of vertices where  $\phi_i(v) \neq 0$  and let  $m$  be the number of interior vertices in  $T_h$ , any function  $\alpha \in C_h$  has a unique representation written as:

$$u(v) = \sum_{i=1}^m u_i \phi_i(v) \quad , v \in \Omega$$

where  $u_i = u(v_i)$ .

The finite element solution is written as:

$$u = u_h(t, x) = \sum_{i=1}^N u_i(t) \phi_i(x) \quad (1.36)$$

where  $\phi_i(x)$  is standard continuous piecewise linear function in space (see [18]).

Substituting the approximation (1.36) into equation (1.35), we get

$$\begin{aligned}
& \sum_{i=1}^N u_i \int_{\Omega} c^2 \frac{d\phi_i}{dx} \cdot \frac{d\phi_j}{dx} dx + \sum_{i=1}^{32} \partial_t^2 u \int_{\Omega} \phi_i(x) \phi_j(x) dx \\
& - \sum_{i=1}^N u_i \int_{\Omega} \phi_j(x) \frac{d\phi_i}{dx} \cdot n d\sigma = \int_{\Omega} f \phi_j(x) dx \quad , \\
& \forall \phi_j(x) \in V_h
\end{aligned} \tag{1.37}$$

Let  $u = u_i$  denote the vector of unknown coefficients,

$A = (a_{ij})$  is mass matrix  $M \times M$  in space, with coefficients

$$A_{ij} = \int_{\Omega} \phi_i(x) \phi_j(x) dx$$

$M = (m_{ij})$  is stiffness matrix  $M \times M$  in space and  $B = (b_{ij})$  , with coefficients

$$m_{ij} = \int_{\Omega} \frac{d\phi_i}{dx} \cdot \frac{d\phi_j}{dx} dx$$

$$b_{ij} = \int_{\Omega} \phi_j(x) \frac{d\phi_i}{dx} \cdot n d\sigma$$

and

$$F = (f_j), f_j = (f, \phi_j),$$

Which can be written in a matrix form as

$$c^2 M u_i + A \partial_t^2 u - B u_i = F$$

Using a finite difference approximation for the time derivative gives

$$(c^2M - B)u_i + A \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta t)^2} = F$$

So

$$u_{i+1} = 2u_i - u_{i-1} - (\Delta t)^2 A^{-1}(c^2M - B)u_i + (\Delta t)^2 A^{-1}F, \\ \forall i = 1, 2, \dots, N - 1. \quad (1.38)$$

### 1.6 Finite Difference Method for Two Dimensional Wave Equation

consider the following two dimensional inhomogeneous wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + G(x, y, t) \quad (1.39)$$

or we can simply write this equation in another form as

$$u_{tt} = c^2 (u_{xx} + u_{yy}) + G \quad (1.40)$$

For a fixed time  $t$ , the surface  $z = u(x, y, t)$ ,  $0 < x < a$ ,  $0 < y < b$

and  $u(x, y, t) = g(x, y, t)$  for any  $(x, y, t) \in B$ , where  $B$  denotes the boundary of a region  $D$ ,  $G(x, y, t)$  is a continuous function on  $D$  and  $g(x, y, t)$  is continuous on  $B$ . The continuity of both  $G$  and  $g$  guarantees a unique solution of equation (1.40).

Two dimensional wave equations are easily discretized by assembling building blocks for discretization of one dimensional wave equations, because the two dimensional versions just contain terms of the same type that occurs in one dimension.

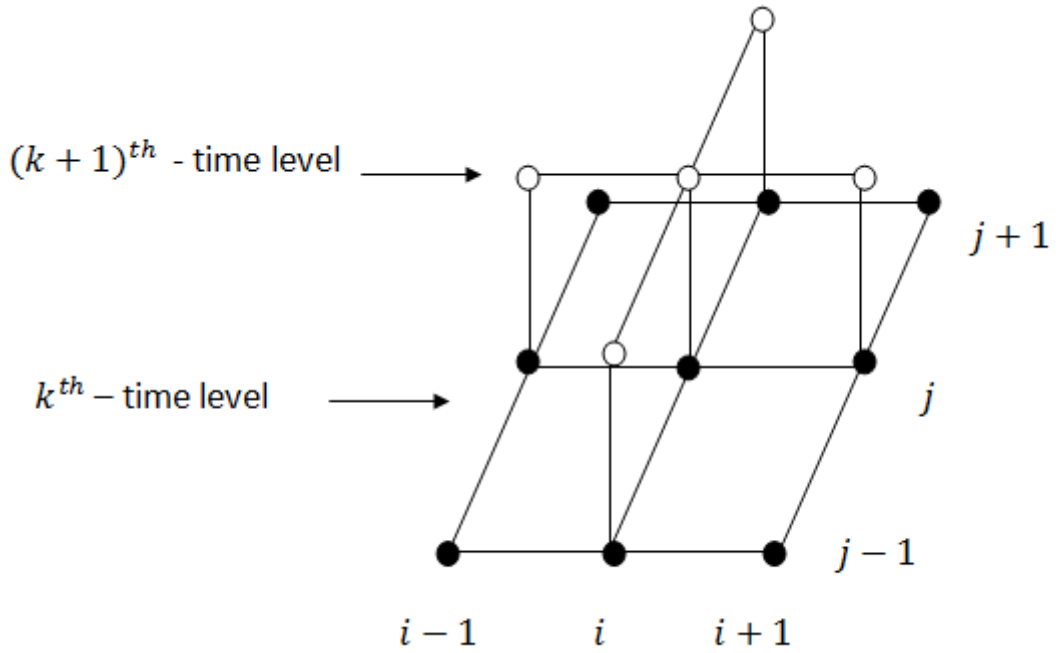
Now, we will use the finite difference algorithm for solving two dimensional wave equation, as shown in figure (1.13) (see [19],[28]).

We introduce a mesh in time and in space. The mesh in time consists of time points  $t_0 = 0 < t < t_k$ , often with a constant spacing  $\Delta t = t_{k+1} - t_k$ .

It is a very common choice to use constant mesh spacings:

$$\Delta x = x_{i+1} - x_i, \quad \Delta y = y_{j+1} - y_j$$

We consider equal mesh spacings such that  $h = \Delta x = \Delta y$ .



**Figure1. 13:** discretization of two dimensional domain

The unknown  $u$  at mesh point  $(x_i, y_j, t_k)$  is denoted by  $u(x_i, y_j, t_k)$  or  $u_{i,j,k}$ .

The difference method is obtained using the centered-difference quotient for the second partial derivatives given by (see [27]):

$$u_{tt} = \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta t)^2} - \frac{(\Delta t)^2}{12} \frac{\partial^4 u(x_i, y_j, \xi_k)}{\partial t^4},$$

where  $\xi_k \in (t_{k-1}, t_{k+1})$  (1.41)

$$u_{xx} = \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h^2} - \frac{h^2}{12} \frac{\partial^4 u(\mu_i, y_j, t_k)}{\partial x^4},$$

where  $\mu_i \in (x_{i-1}, x_{i+1})$  (1.42)

and

$$u_{yy} = \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{h^2} - \frac{h^2}{12} \frac{\partial^4 u(x_i, \eta_j, t_k)}{\partial y^4},$$

where  $\eta_j \in (y_{j-1}, y_{j+1})$  (1.43)

Substituting equations (1.41), (1.42) and (1.43) into equation (1.39) gives

$$\begin{aligned} & \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta t)^2} \\ &= c^2 \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h^2} \\ &+ c^2 \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{h^2} \\ &- \frac{1}{12} \left[ c^2 h^2 \frac{\partial^4 u(\mu_i, y_j, t_k)}{\partial x^4} + c^2 h^2 \frac{\partial^4 u(x_i, \eta_j, t_k)}{\partial y^4} \right. \\ &\left. - (\Delta t)^2 \frac{\partial^4 u(x_i, y_j, \xi_k)}{\partial t^4} \right] + G_{i,j,k} \end{aligned} \quad (1.44)$$

for each  $i = 1, 2, \dots, m-1, j = 1, 2, \dots, r-1$  and  $k = 1, 2, \dots, n-1$ .

Neglecting the error term  $O(h^2 + (\Delta t)^2)$ :

$$\left[ c^2 h^2 \frac{\partial^4 u}{\partial x^4}(\mu_i, y_j, t_k) + c^2 h^2 \frac{\partial^4 u}{\partial x^4}(x_i, \eta_j, t_k) - (\Delta t)^2 \frac{\partial^4 u}{\partial x^4}(x_i, y_j, \xi_k) \right]$$

and multiplying both sides by  $(\Delta t)^2$ , we get:

$$\begin{aligned} u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1} \\ = c^2(\Delta t)^2 \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h^2} \\ + c^2(\Delta t)^2 \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{h^2} + (\Delta t)^2 G_{i,j,k} \end{aligned}$$

Define  $s^2 = c^2(\Delta t)^2/h^2$ , we get :

$$\begin{aligned} u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1} \\ = s^2(u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k} + u_{i,j+1,k} - 2u_{i,j,k} \\ + u_{i,j-1,k}) + (\Delta t)^2 G_{i,j,k} \end{aligned} \quad (1.45)$$

then Simplifying equation (1.45), hence it can be written as:

$$\begin{aligned} 2(2s^2 - 1)u_{i,j,k} \\ = s^2(u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k}) - u_{i,j,k+1} \\ - u_{i,j,k-1} + (\Delta t)^2 G_{i,j,k} \end{aligned} \quad (1.46)$$

For  $i = 1, 2, \dots, m-1, j = 1, 2, \dots, r-1, k = 1, 2, \dots, n-1$ .

with boundary conditions:

1.  $u_{0,j,k} = g(x_0, y_j, t_k)$ , for  $j = 0, 1, 2, \dots, r$  and  $k = 0, 1, 2, \dots, n$ .
2.  $u_{m,j,k} = g(x_m, y_j, t_k)$ , for  $j = 0, 1, 2, \dots, r$  and  $k = 0, 1, 2, \dots, n$ .
3.  $u_{i,0,k} = g(x_i, y_0, t_k)$ , for  $i = 1, 2, \dots, m-1$  and  $k = 1, 2, \dots, n-1$ .
4.  $u_{i,r,k} = g(x_i, y_r, t_k)$ , for  $i = 1, 2, \dots, m-1$  and  $k = 0, 1, 2, \dots, n-1$ .

5.  $u_{i,j,n}=g(x_i, y_j, t_n)$ , for  $i = 0, 1, 2, \dots, m-1$  and  $j = 0, 1, 2, \dots, r-1$ .
6.  $u_{i,j,0}=g(x_i, y_j, t_0)$ , for  $i = 1, 2, \dots, m$  and  $j = 1, 2, \dots, r$ . (1.47)

Equation (1.46) involves approximations to the unknown function  $u(x, y, t)$  at the points  $(x_i, y_j, t_{n+1}), (x_i, y_j, t_n), (x_{i+1}, y_j, t_n), (x_{i-1}, y_j, t_n), (x_i, y_{j+1}, t_n), (x_i, y_{j-1}, t_n), (x_i, y_j, t_{n-1})$ .

When we use formula (1.46) with boundary conditions (1.47), then at all points  $(x_i, y_j, t_n)$  that are adjacent to a boundary mesh point, we have an  $(m-1) \times (r-1) \times (n-1)$  by  $(m-1) \times (r-1) \times (n-1)$  linear system with the unknowns being the approximations  $u_{i,j,k}$  to  $u(x_i, y_j, t_k)$  at the interior mesh points (see [28]).

The generated linear system should be solved by Jacobi, Gauss-Seidel, Successive over Relaxation (SOR), or Conjugate Gradient methods.

### 1.6.1 Two Dimensional Wave Equation with Dirichlet Boundary Conditions

When the function is defined on any part of a domain  $D$ , then we call this part Dirichlet boundary  $S_D$ , i.e. the unknown function  $u$  is prescribed on the boundary, that is,  $u(x, y, t) = g(x, y, t)$ ,

$(x, y, t) \in B$  where the function  $g$  is a known function .

$$B_D: u = g$$

To derive the formula of finite difference approximation with Dirichlet boundary condition for two dimensional wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + G(x, y, t)$$

we consider three points  $i + 1, i$ , and  $i - 1$  which are located on  $x$  -axis with equal distance  $h$  between them, then the value of the function  $u(x, y, t)$  at the points  $(i - 1, j, k)$ ,  $(i, j, k)$ , and  $(i + 1, j, k)$  be  $u_{i-1,j,k}$ ,  $u_{i,j,k}$ , and  $u_{i+1,j,k}$ , respectively (see [19],[27]and [28]).

Now, use Taylor series to express  $u_{i-1,j,k}$  and  $u_{i+1,j,k}$  in the form of Taylor expansions about the point  $i$  as follows:

$$u_{i+1,j,k} = u_{i,j,k} + \frac{h}{1!} \cdot \frac{\partial u}{\partial x} \Big|_i + \frac{h^2}{2!} \cdot \frac{\partial^2 u}{\partial x^2} \Big|_i + \frac{h^3}{3!} \cdot \frac{\partial^3 u}{\partial x^3} \Big|_i + O(h^4) \quad (1.48)$$

$$u_{i-1,j,k} = u_{i,j,k} - \frac{h}{1!} \cdot \frac{\partial u}{\partial x} \Big|_i + \frac{h^2}{2!} \cdot \frac{\partial^2 u}{\partial x^2} \Big|_i - \frac{h^3}{3!} \cdot \frac{\partial^3 u}{\partial x^3} \Big|_i + O(h^4) \quad (1.49)$$

By adding the two equations (1.48) and (1.49), we get:

$$u_{i+1,j,k} + u_{i-1,j,k} = 2u_{i,j,k} + h^2 \cdot \frac{\partial^2 u}{\partial x^2} \Big|_i + O(h^4)$$

By rearranging the above equation, we get:

$$\frac{\partial^2 u}{\partial x^2} \Big|_i = \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h^2} + O(h^2) \quad (1.50)$$

Equation (1.50) is a finite difference approximation formula with the error term  $O(h^2)$ .

Now, subtracting equation (1.49) from equation (1. 48), we get:



$$\frac{\partial u}{\partial x} \Big|_i = \frac{u_{i+1,j,k} - u_{i-1,j,k}}{2h} + O(h^2) \quad (1.51)$$

Equation (1.51) is a finite difference approximation formula with the error term  $O(h^2)$ .

Similarly, consider three points  $j - 1, j$ , and  $j + 1$  which are located on the  $y$  -axis with equal distance  $h$  between them. Let the value of the function  $u(x, y, t)$  at the points  $(i, j - 1, k)$ ,  $(i, j, k)$ , and  $(i, j + 1, k)$  be  $u_{i,j-1,k}$ ,  $u_{i,j,k}$  and  $u_{i,j+1,k}$ , respectively.

Use Taylor series to express  $u_{i,j-1,k}$  and  $u_{i,j+1,k}$  in the form of Taylor expansions about the point  $j$ , the finite difference approximation formulas with the error term  $O(h^2)$  of second order for  $\frac{\partial^2 u}{\partial y^2} \Big|_j$  and  $\frac{\partial u}{\partial y} \Big|_j$  are, respectively:

$$\frac{\partial^2 u}{\partial y^2} \Big|_j = \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{2h} + O(h^2) \quad (1.52)$$

and

$$\frac{\partial u}{\partial y} \Big|_j = \frac{u_{i,j+1,k} - u_{i,j-1,k}}{2h} + O(h^2) \quad (1.53)$$

Similarly, consider three points  $k - 1, k$ , and  $k + 1$  which are located on the  $t$  -axis with equal distance  $h$  between them. Let the value of the function  $u(x, y, t)$  at the points  $(i, j, k - 1)$ ,  $(i, j, k)$ , and  $(i, j, k + 1)$  be  $u_{i,j,k-1}$ ,  $u_{i,j,k}$  and  $u_{i,j,k+1}$ , respectively.

Now, use Taylor series to express  $u_{i,j,k-1}$  and  $u_{i,j,k+1}$  in the form of Taylor expansions about the point  $n$ , the finite difference

approximation formula with the error term  $O(h^2)$  of second order for  $\frac{\partial^2 u}{\partial t^2}|_n$  is

$$u_{tt} = \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta t)^2} \quad (1.54)$$

Inserting equations (1.50), (1.54) and (1.52) into equation (1.39)

yields:

$$\begin{aligned} & \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta t)^2} \\ &= c^2 \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h^2} \\ &+ c^2 \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{h^2} + G(x, y, t) \end{aligned} \quad (1.55)$$

by rearranging equation (1.55) and let  $s^2 = c^2(\Delta t)^2/h^2$ , , we get

$$\begin{aligned} & 2(2s^2 - 1)u_{i,j,k} \\ &= s^2(u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k}) - u_{i,j,k+1} \\ &- u_{i,j,k-1} + (\Delta t)^2 G_{i,j,k} \end{aligned} \quad (1.56)$$

In general, if  $u$  satisfies the wave equation, then  $u$  at any point in the domain  $D$  satisfies the above equation .

Now, suppose we have Dirichlet boundary conditions defined on the box-shape domain such that  $1 \leq i \leq m$ ,  $1 \leq j \leq r$  and  $1 \leq k \leq n$  .

Let  $u(x, y, t) = g(x, y, t)$  be given on all boundaries of the domain, that is  $u = g$  is defined on the all boundaries.

In other words, the values of the points  $(x_i, y_j, t_k)$ ,  $\forall i = 2, 3, \dots, m-1$ ,  $\forall j = 2, 3, \dots, r-1$  and  $\forall k = 2, 3, \dots, n-1$  under the function  $g$  are

known. For the corner grid points, we use the following equations (see [28]) :

$$\begin{aligned}
u(1,1,1) &= \frac{1}{3}[u(2,1,1) + u(1,2,1) + u(1,1,2)] \\
u(1,1,n) &= \frac{1}{3}[u(2,1,n) + u(1,2,n) + u(1,1,n-1)] \\
u(1,r,1) &= \frac{1}{3}[u(1,r-1,1) + u(2,r,1) + u(1,r,2)] \\
u(m,1,1) &= \frac{1}{3}[u(m-1,1,1) + u(m,1,2) + u(m,2,1)] \\
u(m,1,n) &= \frac{1}{3}[u(m,1,n-1) + u(m-1,1,n) + u(m,2,n)] \\
u(1,r,n) &= \frac{1}{3}[u(1,r,n-1) + u(2,r,n) + u(1,r-1,n)] \\
u(m,r,n) &= \frac{1}{3}[u(m,r,n-1) + u(m,r-1,n) + u(m-1,r,n)] . \\
u(m,r,1) &= \frac{1}{3}[u(m,r-1,1) + u(m-1,r,1) + u(m,r,2)] \quad (1.42).
\end{aligned}$$

### 1.6.2 Two Dimensional Wave Equation with Neumann Boundary Conditions

When the normal derivative of the unknown function  $u$  is prescribed on the boundary of a domain  $D$ , then we call this part Neumann boundary  $B_N$ , i.e. the value of the normal derivative  $\frac{\partial u}{\partial n} = g(x, y, t)$  is given on the boundary of the domain, where  $g(x, y, t)$  is a given function.

$$B_N: \frac{\partial u}{\partial n} = g(x, y, t)$$

We derive the formula of finite difference approximation with Neumann boundary condition for two dimensional wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + G(x, y, t)$$

Suppose that Dirichlet condition is specified on up, right and down walls and Neumann condition is defined on the remaining wall which is the left wall as follows:

$$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial x} = 0 \quad (1.58)$$

Now, we want to approximate equation (1.24) using the second order approximation using equation (1.16). This procedure puts the grid points  $(1, j, k)$  outside the domain towards the left that is located on imaginary boundary that their fake coordinates will be  $(0, j, k)$ .

(see [21],[27]). So, equation (1.58) is approximated using equation (1.53) at the line  $i = 1$ .

$$\frac{\partial u}{\partial y} \Big|_{i=1} = \frac{u_{1+1,j,k} - u_{1-1,j,k}}{2h} = 0$$

From this it follows that  $u_{2,j,k} = u_{0,j,k}$ . The discretized wave equation at the boundary point  $(1, j, n)$  reads

$$\begin{aligned} & 2(1 - 2s^2)u_{1,j,k} \\ &= s^2(u_{2,j,k} + u_{0,j,k} + u_{1,j+1,k} + u_{1,j-1,k}) - u_{1,j,k+1} \\ & - u_{1,j,k-1} + (\Delta t)^2 G_{1,j,k} \end{aligned} \quad (1.59)$$

We can then just insert  $u_{2,j,k}$  for  $u_{0,j,k}$  into equation (1.59) and then solve for the boundary value  $u_{1,j,k}$ . So we get,

$$2(1 - 2s^2)u_{1,j,k} = s^2(2u_{2,j,k} + u_{1,j+1,k} + u_{1,j-1,k}) - u_{1,j,k+1} - u_{1,j,k-1} + (\Delta t)^2 G_{1,j,k} \quad (1.60)$$

For any positive integers  $m, r$  and  $n$ , we use equation (1.44) for

$2 \leq j \leq r - 1$  and  $2 \leq k \leq n - 1$ . As Dirichlet conditions are specified on other boundaries, the values are known. To find the corner boundary values, we use equation (1.57) (see [27]).

## Chapter Two

### Iterative Methods for Solving Linear Systems

In chapter one, a linear system was generated using the finite difference method (FDM) and the finite element method (FEM) to describe the partial differential equations that can be solved by iterative techniques. In this chapter, we will solve such linear systems by iterative methods and discuss their convergence properties (see [31]).

For solving an  $n \times n$  linear system

$$A\mathbf{x} = \mathbf{b}$$

We start with an initial approximation  $\mathbf{x}^{(0)}$  to the solution  $\mathbf{x}$  and then generate a sequence  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  that converges to  $\mathbf{x}$ .

Most iterative techniques involve a process of converting the system

$A\mathbf{x} = \mathbf{b}$  into an equivalent system:

$$\mathbf{x} = T\mathbf{x} + \mathbf{c}$$

where  $T$  is an  $n \times n$  matrix and  $\mathbf{c}$  is a column vector.

After selecting an initial approximation  $\mathbf{x}^{(0)}$ , we generate a sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  defined as:

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}, \quad k \geq 1$$

The used iterative methods here are :

1. Jacobi Method.
2. Gauss-Seidel Method.
3. Successive over Relaxation (SOR) Method.
4. Conjugate Gradient Method.

consider the  $n \times n$  (square) linear system:

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\
 a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n
 \end{aligned} \tag{2.1}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

We can simply write this system in matrix form as:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Then, we can convert system (2.1) into the form:

$$\mathbf{x} = T \mathbf{x} + \mathbf{c}$$

Then selecting an initial approximation  $\mathbf{x}^{(0)}$ , we generate a sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  defined as (see [5]):

$$\mathbf{x}^{(k)} = T \mathbf{x}^{(k-1)} + \mathbf{c}, \quad k \geq 1$$

## 2.1 Jacobi Method

The Jacobi method is an algorithm in linear algebra for determining the solutions of a system of linear equations with largest absolute values in each row and column dominated by the diagonal element. Each diagonal element is solved and an approximate value plugged in. This process is then iterated until it converges.

The Jacobi method is the simplest iterative method for solving a (square) linear system  $A\mathbf{x} = \mathbf{b}$ . (see [5] ).

### The General Formula of Jacobi Method

In general, the Jacobi iterative method is given by the sequence :

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[ \sum_{j=1}^n -a_{ij}x_j^{(k-1)} + b_i \right], j \neq i, a_{ii} \neq 0,$$

$$\text{for } i = 1, 2, \dots, n, k \in \mathbb{N}^* \quad (2.2)$$

We can derive formula (2.2) by splitting matrix  $A$  into its diagonal and off-diagonal parts (see [31],[5]).

Let  $D$  be the diagonal matrix where entries are those of matrix  $A$ , let  $-L$  be the strictly lower triangular part of matrix  $A$  and let  $-U$  be the strictly upper triangular part of matrix  $A$ . With this notation matrix  $A$  is split into:

$$A = D - L - U \quad (2.3)$$



where

$$D = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}, L = \begin{bmatrix} 0 & 0 & \dots & 0 \\ -a_{21} & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \dots & 0 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 0 & -a_{12} & \dots & -a_{1n} \\ 0 & 0 & \dots & -a_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

By substituting formula (2.2) into  $A\mathbf{x} = \mathbf{b}$ , we get:

$$(D - L - U)\mathbf{x} = \mathbf{b}$$

The above equation can be written as:

$$D\mathbf{x} = (L + U)\mathbf{x} + \mathbf{b}$$

If  $D^{-1}$  exists, then:

$$\mathbf{x} = D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}$$

This result is the matrix form of the Jacobi scheme:

$$\mathbf{x}^{(k)} = D^{-1}(L + U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b}$$

Using  $T_j = D^{-1}(L + U)$  and  $c_j = D^{-1}\mathbf{b}$ , we obtain the Jacobi technique of the form:

$$\mathbf{x}^{(k)} = T_j \mathbf{x}^{(k-1)} + c_j, \quad k \geq 1$$

So,

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[ \sum_{j=1}^n -a_{ij}x_j^{(k-1)} + b_i \right], j \neq i, a_{ii} \neq 0, i = 1, 2, \dots, n.$$

Conclusion: to find  $\mathbf{x}^{(k)}$  approximation we must know  $\mathbf{x}^{(k-1)}$  approximation for any  $k \geq 1$  where  $k \in \mathbb{N}$ . Continuing this procedure, we obtain a sequence of approximations (see [5] and [29]).

## 2.2 Gauss-Seidel Method

This iterative method is used for solving a square linear system  $A\mathbf{x} = \mathbf{b}$ , it is similar to the Jacobi method.

But with the Jacobi method, the values of  $x_i^{(k)}$  obtained in the  $k^{th}$  iteration remain unchanged until the entire  $(k+1)^{th}$  iteration has been calculated. With the Gauss-Seidel method, we use the new values  $x_i^{(k+1)}$  as soon as they are known. For example, once we have computed  $x_1^{(k+1)}$  from the first equation, its value is then used in the second equation to obtain the new  $x_2^{(k+1)}$  and so on, this is the difference between the Jacobi and Gauss-Seidel methods (see [5]).

### The General Formula of Gauss-Seidel Method

In general, the Gauss-Seidel iterative method is given by the sequence

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[ - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} + b_i \right], a_{ii} \neq 0,$$

$$\text{for } i = 1, 2, \dots, n, k \in \mathbb{N}^* \quad (2.4)$$

We can derive formula (2.4) by substituting formula (2.3) into  $A\mathbf{x} = \mathbf{b}$ , so we get (see [5],[31]):

$$(D - L - U)\mathbf{x} = \mathbf{b}$$

The above equation can be written as:

$$(D - L)\mathbf{x} = U\mathbf{x} + \mathbf{b}$$

If  $(D - L)^{-1}$  exists, then:

$$\mathbf{x} = (D - L)^{-1} U\mathbf{x} + (D - L)^{-1}\mathbf{b}$$

This result is the matrix form of the Gauss-Seidel scheme:

$$\mathbf{x}^{(k)} = (D - L)^{-1} U\mathbf{x}^{(k-1)} + (D - L)^{-1}\mathbf{b}$$

Using  $T_g = (D - L)^{-1}U$  and  $c_g = (D - L)^{-1}\mathbf{b}$ , we obtain the Gauss-Seidel technique of the form:

$$\mathbf{x}^{(k)} = T_g \mathbf{x}^{(k-1)} + c_g, \quad k \geq 1.$$

### 2.3 Successive over Relaxation Method (SOR Method)

The main constraint in using this method is that the coefficient matrix of the linear system  $A\mathbf{x} = \mathbf{b}$  must be symmetric and positive definite. For any positive real number called the relaxation parameter (factor),  $\omega \in (0,2)$ . When  $0 < \omega < 1$ , the method is called Successive under Relaxation and can be used to achieve convergence for systems that are not convergent by the Gauss-Seidel method. However, if

$1 < \omega < 2$ , then the method is called Successive over Relaxation method. If  $\omega = 1$ , then we get Gauss-Seidel method (see [5],[14] ).

### The General Formula of SOR Method

The derivation of the general formula of SOR method depends on Gauss-Seidel formula. Consider Gauss-Seidel formula, that is (2.4):

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[ - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)} + b_i \right], a_{ii} \neq 0$$

Defining the difference:

$$\Delta x_i = x_i^{(k)} - x_i^{(k-1)}$$

This can be written as:

$$x_i^{(k)} = x_i^{(k-1)} + \Delta x_i$$

Now, multiplying  $\Delta x_i$  by the relaxation parameter  $\omega$  in the last expression, we get:

$$\begin{aligned} x_i^{(k)} &= x_i^{(k-1)} + \omega \Delta x_i \\ &= x_i^{(k-1)} + \omega (x_i^{(k)} - x_i^{(k-1)}) \end{aligned}$$

So, 
$$x_i^{(k)} = (1 - \omega) x_i^{(k-1)} + \omega x_i^{(k)}$$

Substituting the Gauss-Seidel formula (2.4) into the last expression, we get:

$$x_i^{(k)} = (1 - \omega) x_i^{(k-1)} + \frac{\omega}{a_{ii}} \left[ - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)} + b_i \right],$$

$$a_{ii} \neq 0, i = 1, 2, 3, \dots, n. \quad (2.5)$$

Formula (2.5) is called the SOR iterative method (see [5] ).

We can express formula (2.5) in matrix form as follows (see [5], [31] ):

Since  $a_{ii} \neq 0$ , then we can multiply formula (2.5) by  $a_{ii}$  to get:

$$a_{ii}x_i^{(k)} = a_{ii}(1 - \omega)x_i^{(k-1)} + \omega \left[ - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} + b_i \right]$$

Simplifying the last formula, we get

$$a_{ii}x_i^{(k)} = (1 - \omega)a_{ii}x_i^{(k-1)} - \omega \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \omega \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} + \omega b$$

By rearranging the above equation, we get

$$\begin{aligned} a_{ii}x_i^{(k)} + \omega \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} &= (1 - \omega)a_{ii}x_i^{(k-1)} - \omega \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} + \omega b \end{aligned}$$

Then,

$$(D - \omega L)\mathbf{x}^{(k)} = ((1 - \omega)D + \omega U)\mathbf{x}^{(k-1)} + \omega \mathbf{b}$$

Now, if  $(D - \omega L)^{-1}$  exists, then we have:

$$\mathbf{x}^{(k)} = (D - \omega L)^{-1} ((1 - \omega)D + \omega U)\mathbf{x}^{(k-1)} + \omega(D - \omega L)^{-1} \mathbf{b}$$

Then, we get the matrix form of SOR method as:

$$\mathbf{x}^{(k)} = T_\omega \mathbf{x}^{(k-1)} + c_\omega, \quad k \geq 1,$$

Where:  $T_\omega = (D - \omega L)^{-1} ((1 - \omega)D + \omega U)$  and  $c_\omega = \omega(D - \omega L)^{-1} \mathbf{b}$ .

## 2.4 Conjugate Gradient Method

The conjugate gradient method is a numerical iterative method used to approximate the exact solution of particular linear system  $A\mathbf{x} = \mathbf{b}$  where the coefficient matrix  $A$  must be symmetric and positive definite (see [25]).

### General Formulas Needed to Compute The Conjugate Gradient Method Algorithm

Suppose we want to solve the following  $n \times n$  linear system:

$$A\mathbf{x} = \mathbf{b}$$

Where  $A$  is symmetric and positive definite matrix,  $\mathbf{x}$  and  $\mathbf{b}$  are column vectors ( $n \times 1$  – matrices).

The solution of  $A\mathbf{x} = \mathbf{b}$  uniquely minimizes the following quadratic form:

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$$

Suppose that  $p$  is a basis of  $\mathbb{R}^n$  where:

$p = \{ p_k | p_i \cdot p_k = 0 \text{ with respect to matrix } A, \forall i \neq k, \text{ where } 1 \leq i, k \leq n \}$  is a set of  $n$  mutually conjugate (orthogonal) directions.

We will write the conjugate gradient iterative method algorithm as follows (see [25]):

**Step 1:** Start with initial guess  $\mathbf{x}_0$  that may be considered 0 if otherwise is not given.

**Step 2:** Calculate the residual vector  $\mathbf{r}_0$  as follows:

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$$

**Step 3:** Let the initial direction vector  $\mathbf{p}_0 = \mathbf{r}_0$ , that is, the negative of the gradient of the quadratic function:

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x} \text{ at } \mathbf{x} = \mathbf{x}_0.$$

Note that  $\mathbf{p}_k$  will change in each iteration.

**Step 4:** Compute the scalars  $\alpha_k$ 's using the formula:

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T A \mathbf{p}_k}, \forall k = 0, 1, 2, \dots, n-1.$$

**Step 5:** Compute the first iteration  $\mathbf{x}_1$  using the formula:

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{p}_0$$

**Step 6:** Compute the residual vectors  $\mathbf{r}_k$ 's using the formula:

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A \mathbf{p}_k, \forall k = 0, 1, 2, \dots, n-1.$$

**Step 7:** Compute the scalars  $\beta_k$ 's using the formula:

$$\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}, \forall k = 0, 1, 2, \dots, n-1.$$

**Step 8:** Compute the direction vectors  $\mathbf{p}_k$ 's using the formula:

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k, \forall k = 0, 1, 2, \dots, n-1.$$

**Step 9:** Compute the iterations  $\mathbf{x}_k$  using the formula ( see [25],[31],[39]):

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \forall k = 1, 2, \dots, n - 1.$$

## 2.5 Convergence of Iterative Methods

In this section, the general aim is to study the convergence for each previous iterative method and then make a comparison between them. After that, we will conclude the fastest method. In any computational problem, we get high accuracy if the error becomes very small. In our iterative methods problem, the actual error  $e$  is the difference between the exact solution  $x$  and the approximate solution  $\mathbf{x}^{(k)}$ . But we cannot compute its value since we do not know the exact solution. Instead of that, we will deal with the estimate error which is equal the difference between the approximate solution  $\mathbf{x}^{(k)}$  and the next approximate solution  $\mathbf{x}^{(k+1)}$  ( see [5] ).

Therefore, we can compute more iterations with less errors and hence we get high level of accuracy.

Suppose  $\mathbf{x}$  is the exact solution of the following linear system:

$$A\mathbf{x} = \mathbf{b} \tag{2.6}$$

This can be written in equivalent form as:

$$\mathbf{x}^{(k)} = T \mathbf{x}^{(k-1)} + \mathbf{c}, \quad k \geq 1 \tag{2.7}$$

where  $T$  is an  $n \times n$  matrix and  $\mathbf{c}$  is a column vector.



The idea of the iterative methods is to generate a sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  that converges to the exact solution  $\mathbf{x}$  of the linear system (2.6). (Note that each vector in the sequence is an approximation to the exact solution) (see [14] ).

### 2.5.1 Convergence of Jacobi and Gauss-Seidel Iterative Methods

The following theorems hold for Jacobi and Gauss-Seidel iterative methods :

**Theorem 2.1** (see [39])

For any initial approximation, a sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  converges to the exact solution  $\mathbf{x}$  if and only if the spectral radius of the square matrix  $T$ ,  $\rho(T) < 1$ . ( $T$  is the matrix as in (2.7) form).

**Theorem 2.2** (see [39])

If the coefficient matrix  $A$  for the linear system (2.6) is strictly diagonally dominant, then the sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  generated by the Jacobi and Gauss-Seidel Iterative techniques converges to the unique solution of that system .

**Theorem 2.3** (see [5])

If  $\|T\| < 1$  (any norm of  $T$ ) then the sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  converges to a vector  $\mathbf{x} \in \mathbb{R}^n$  for any initial approximation vector  $\mathbf{x}^{(0)} \in \mathbb{R}^n$ .

### 2.5.2 Convergence of SOR iterative Method

**Theorem 2.4** (see [39])

Theorem 2.1 holds for SOR method.

**Theorem 2.5 “ Ostrowski-Reich ”** (see [5])

If the coefficient matrix  $A$  of the linear system (2.6) is a positive definite matrix and the relaxation parameter (factor)  $\omega \in (0,2)$ , then the SOR method converges for any choice of initial approximation vector  $\mathbf{x}^{(0)}$ .

### 2.5.3 Convergence of Conjugate Gradient Method

**Theorem 2.6** (see [25])

The sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  generated by the Conjugate Gradient method converges to the solution  $\mathbf{x}$  of the square linear system  $A\mathbf{x} = \mathbf{b}$  of  $n$  variables in at most  $n$  steps for any choice of initial approximation vector  $\mathbf{x}^{(0)}$ .

**Proof:** (see [25]).

suppose  $\mathbf{x}$  is the exact solution and  $\mathbf{x}^{(0)}$  is the initial solution.

The set of direction vectors are orthogonal so they are linearly independent. Therefore, they span the space  $\mathbb{R}^n$ . Hence, we can write:

$$\mathbf{x} - \mathbf{x}^{(0)} = a_0 \mathbf{p}_0 + a_1 \mathbf{p}_1 + a_2 \mathbf{p}_2 + \cdots + a_{n-1} \mathbf{p}_{n-1}, \text{ where } a_i \text{'s} \in \mathbb{R}.$$

Multiplying both sides of the last expression by  $\mathbf{p}_j^T A$ , we obtain

$$\mathbf{p}_j^T A(\mathbf{x} - \mathbf{x}^{(0)}) = \mathbf{p}_j^T A(a_0 \mathbf{p}_0 + a_1 \mathbf{p}_1 + a_2 \mathbf{p}_2 + \cdots + a_{n-1} \mathbf{p}_{n-1})$$

Simplify the above expression, we get

$$\begin{aligned} \mathbf{p}_j^T A \mathbf{x} - \mathbf{p}_j^T A \mathbf{x}^{(0)} \\ = a_0 \mathbf{p}_j^T A \mathbf{p}_0 + a_1 \mathbf{p}_j^T A \mathbf{p}_1 + a_2 \mathbf{p}_j^T A \mathbf{p}_2 + \cdots + a_{n-1} \mathbf{p}_j^T A \mathbf{p}_{n-1} \end{aligned}$$

but  $\mathbf{b} = A \mathbf{x}$ ,  $\mathbf{r}_0 = \mathbf{b} - A \mathbf{x}_0$ , and  $\mathbf{p}_j^T A \mathbf{p}_i = 0, \forall i \neq j$ . So, it becomes:

$$\mathbf{p}_j^T \mathbf{r}_0 = a_j \mathbf{p}_j^T A \mathbf{p}_j$$

Thus,

$$a_j = \frac{\mathbf{p}_j^T \mathbf{r}_0}{\mathbf{p}_j^T A \mathbf{p}_j} \quad (*)$$

Now, we want to show that  $a_j = \alpha_j$  where

$$\alpha_j = \frac{\mathbf{r}_j^T \mathbf{r}_j}{\mathbf{p}_j^T A \mathbf{p}_j}, \forall j = 0, 1, 2, \dots, n-1.$$

$$\mathbf{x}_j = \mathbf{x}_0 + a_0 \mathbf{p}_0 + a_1 \mathbf{p}_1 + a_2 \mathbf{p}_2 + \cdots + a_{j-1} \mathbf{p}_{j-1}$$

Multiply both sides of the last equation by  $\mathbf{p}_j^T A$

$$\begin{aligned} \mathbf{p}_j^T A \mathbf{x}_j &= \mathbf{p}_j^T A (\mathbf{x}_0 + a_0 \mathbf{p}_0 + a_1 \mathbf{p}_1 + a_2 \mathbf{p}_2 + \cdots + a_{j-1} \mathbf{p}_{j-1}) \\ &= \mathbf{p}_j^T A \mathbf{x}_0 + \mathbf{p}_j^T A (a_0 \mathbf{p}_0 + a_1 \mathbf{p}_1 + a_2 \mathbf{p}_2 + \cdots + a_{j-1} \mathbf{p}_{j-1}) \\ &= \mathbf{p}_j^T A \mathbf{x}_0 + 0 \end{aligned}$$

The above can be written as:

$$\mathbf{p}_j^T A \mathbf{x}_j - \mathbf{p}_j^T A \mathbf{x}_0 = 0$$

Or

$$\mathbf{p}_j^T A (\mathbf{x}_j - \mathbf{x}_0) = 0$$

Therefore,

$$\begin{aligned} \mathbf{p}_j^T \mathbf{r}_0 &= \mathbf{p}_j^T A (\mathbf{x} - \mathbf{x}^{(0)}) \\ &= \mathbf{p}_j^T A (\mathbf{x} - \mathbf{x}_j + \mathbf{x}_j - \mathbf{x}^{(0)}) \end{aligned}$$

$$\begin{aligned}
\mathbf{p}_j^T \mathbf{r}_0 &= \mathbf{p}_j^T A(\mathbf{x} - \mathbf{x}_j) + \mathbf{p}_j^T A(\mathbf{x}_j - \mathbf{x}^{(0)}) \\
&= \mathbf{p}_j^T (A\mathbf{x} - A\mathbf{x}_j) + 0 \\
&= \mathbf{p}_j^T (\mathbf{b} - A\mathbf{x}_j) \\
&= \mathbf{p}_j^T \mathbf{r}_j
\end{aligned}$$

Now, put  $\mathbf{p}_j^T \mathbf{r}_0 = \mathbf{p}_j^T \mathbf{r}_j$  in equation (\*), then we get:

$$a_j = \frac{\mathbf{p}_j^T \mathbf{r}_j}{\mathbf{p}_j^T A \mathbf{p}_j} = \alpha_j$$

This completes the proof.

## Chapter Three

### Numerical Results

In this chapter, we will deal with homogeneous and inhomogeneous one and two dimensional wave equations with Dirichlet and then with Neumann boundary conditions by using finite difference method. Next, we will deal with homogeneous and inhomogeneous one dimensional wave equation with Dirichlet and then with Neumann boundary conditions but by using finite element method. At the end of this chapter, we will make a comparison between the iterative methods that are used for solving the linear system by finite difference and finite element methods.

#### Example 3.1

Consider the following homogeneous one dimensional wave equation

$$4 u_{xx} - u_{tt} = 0$$

with square domain  $D = \{(x, t) | a = 0 < x < b = 1,$

$c = 0 < t < d = 1\}$  subject to Dirichlet boundary conditions given on the boundaries as illustrated in figure 3.1, such that :

$$u(0, t) = 0, u(1, t) = t, u(x, 0) = 0, \text{ and } u(x, 1) = x.$$

we want to approximate the solution  $u$  by using:

- 1- Finite Difference Method (FDM).
- 2- Finite Element Method (FEM).

### 1-Applying Finite Difference Algorithm:

**Step 1:** Choose positive integers  $n = m = 3$ .

**Step 2:** Define  $h = \frac{b-a}{n} = \frac{1-0}{3} = \frac{1}{3}$  and  $k = \frac{d-c}{m} = \frac{1-0}{3} = \frac{1}{3}$

This step partitions the interval  $[0,1]$  on  $x$ -axis into  $n = 3$  equal parts of width  $h = \frac{1}{3}$  and partitions the interval  $[0,1]$  on  $y$ -axis into  $m = 3$  equal parts of width  $k = \frac{1}{3}$  as step 3 illustrates.

**Step 3:** Define the mesh point  $(x_i, t_j)$  as

$$x_i = a + ih, i = 0,1,2, n = 3$$

$$t_j = c + jk, j = 0,1,2, m = 3$$

$$\text{for } i = 0 : x_0 = 0 + (0)\frac{1}{3} = 0 = a$$

$$\text{for } i = 1 : x_1 = 0 + (1)\frac{1}{3} = \frac{1}{3}$$

$$\text{for } i = 2 : x_2 = 0 + (2)\frac{1}{3} = \frac{2}{3}$$

$$\text{for } i = 3 : x_3 = 0 + (3)\frac{1}{3} = 1 = b$$

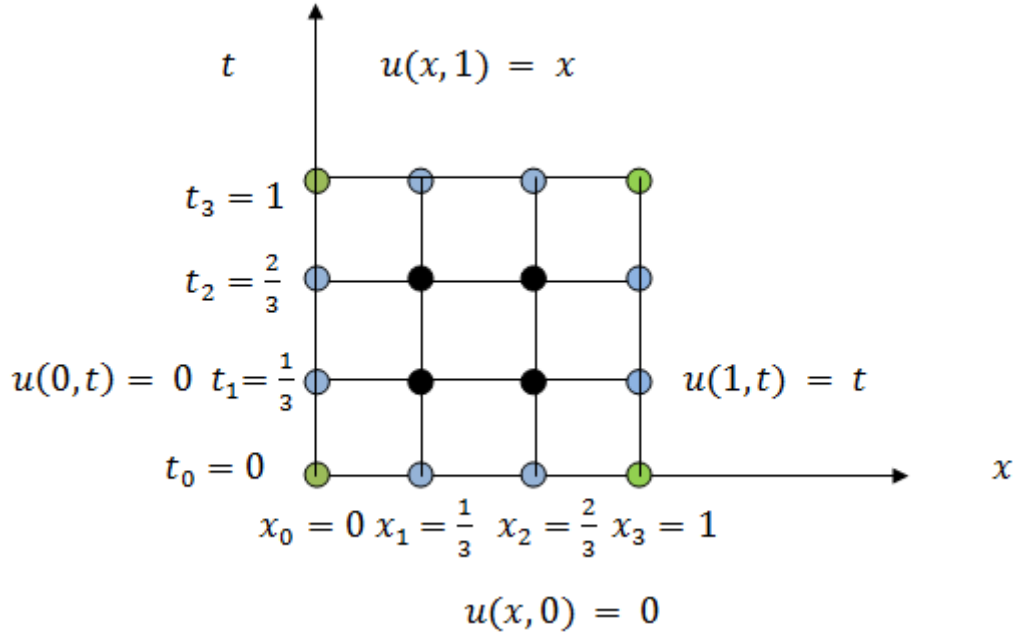
$$\text{for } j = 0 : t_0 = 0 + (0)\frac{1}{3} = 0 = c$$

$$\text{for } j = 1 : t_1 = 0 + (1)\frac{1}{3} = \frac{1}{3}$$

$$\text{for } j = 2 : t_2 = 0 + (2)\frac{1}{3} = \frac{2}{3}$$

$$\text{for } j = 3 : t_3 = 0 + (3)\frac{1}{3} = 1 = d$$

Step 2 and step 3 are illustrated in figure 3.1.



**Figure3. 1:** discretization of the domain for example 1

The blue points are known boundary points and the green points are corner points that are easy to be calculated by equation (1.21). However, the black (interior) points are not known which are to be approximate.

Now, we use the difference equation (1.20) to approximate the interior (black points) mesh points as follows:

$$u_{i,j} = \frac{1}{2(1-c^2)} [u_{i,j+1} + u_{i,j-1} - c^2 u_{i+1,j} - c^2 u_{i-1,j}]$$

For  $i = 1$ , and  $j = 1$  :

$$u_{1,1} = \frac{1}{2(1-4)} [u_{1,1+1} + u_{1,1-1} - 4u_{1+1,1} - 4u_{1-1,1}]$$

$$u_{1,1} = \frac{-1}{6} \times [u_{1,2} + u_{1,0} - 4u_{2,1} - 4u_{0,1}] \quad (3.1)$$

But both  $u_{0,1}$  and  $u_{1,0}$  are known boundary points where  $u_{1,2}$  and  $u_{2,1}$  are unknown. So the value of  $u_{0,1}$  and  $u_{1,0}$  are  $u_{0,1} = 0$  (on the left boundary) and  $u_{1,0} = 0$  (on the down boundary), so equation (3.1) becomes

$$6u_{1,1} = -u_{1,2} + 4u_{2,1}$$

$$6u_{1,1} + u_{1,2} - 4u_{2,1} = 0 \quad (3.2)$$

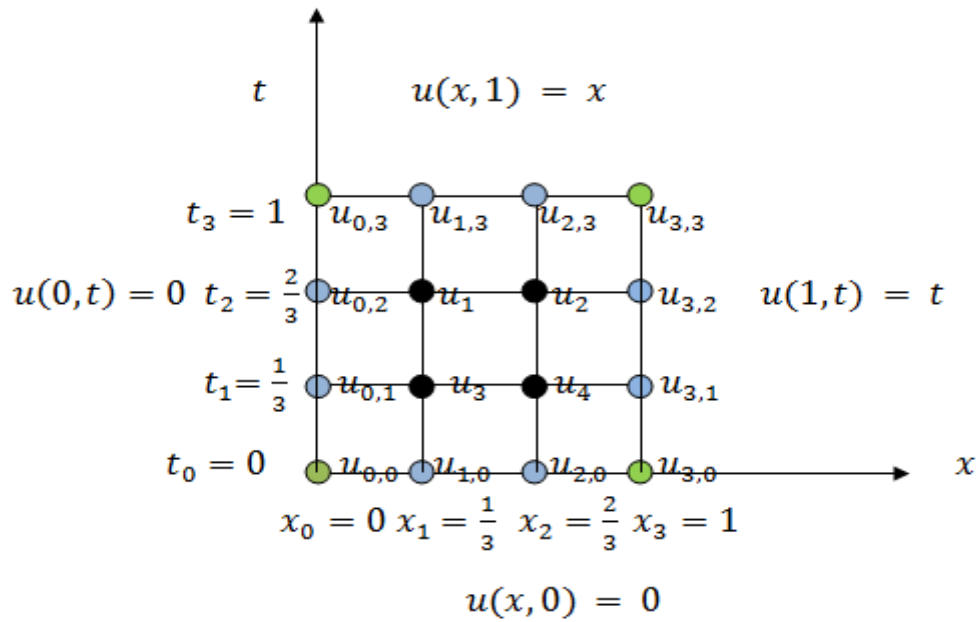
We can label these mesh points as follows:

$$u_r = u_{i,j}$$

$$\text{where } r = i + (m - 1 - j)(n - 1)$$

$$\forall i = 1, 2, 3 \text{ and } \forall j = 1, 2, 3.$$

$$u_{1,1} = u_3, u_{1,2} = u_1, u_{2,1} = u_4, u_{2,2} = u_2$$



**Figure3. 2:** discretization the domain with dirichlet boundary condition for example 1



After labeling the interior mesh points as shown in figure 3.2, then equation (3.2) becomes:

$$u_1 + 6u_3 - 4u_4 = 0 \quad (3.3)$$

In similar manner, we get the following difference equations

$$\text{For } i = 2 \text{ and } j = 1: \quad u_2 - 4u_3 + 6u_4 = \frac{4}{3} \quad (3.4)$$

$$\text{For } i = 1 \text{ and } j = 2: \quad 6u_1 - 4u_2 + u_3 = -\frac{1}{3} \quad (3.5)$$

$$\text{For } i = 2 \text{ and } j = 2: \quad -4u_1 + 6u_2 + u_4 = 2 \quad (3.6)$$

rearrange the equations (3.3),(3.4),(3.5) and (3.6) then we get

$$6u_1 - 4u_2 + u_3 = -\frac{1}{3}$$

$$-4u_1 + 6u_2 + u_4 = 2$$

$$u_1 + 6u_3 - 4u_4 = 0$$

$$u_2 - 4u_3 + 6u_4 = \frac{4}{3}$$

This linear system could be written in matrix form as

$Au = b$ , where  $u$  is a vector of unknowns

$$A = \begin{bmatrix} 6 & -4 & 1 & 0 \\ -4 & 6 & 0 & 1 \\ 1 & 0 & 6 & -4 \\ 0 & 1 & -4 & 6 \end{bmatrix}, b = \begin{bmatrix} -\frac{1}{3} \\ 2 \\ 0 \\ \frac{4}{3} \end{bmatrix}$$

If we apply Gaussian elimination to this linear system, then we get the following exact solution:

$$u = \begin{bmatrix} 0.2222222222222222 \\ 0.4444444444444444 \\ 0.1111111111111111 \\ 0.2222222222222222 \end{bmatrix}$$

We can solve this linear system by any iterative method like Jacobi method, Gauss-Seidel method, Successive over Relaxation (SOR) method and Conjugate Gradient method.

### Jacobi method

It is given by the sequence (2.2):

$$u_i^{(k)} = \frac{1}{a_{ii}} \left[ \sum_{j=1}^n -a_{ij}u_j^{(k-1)} + b_i \right], j \neq i, a_{ii} \neq 0,$$

$$\text{for } i = 1, 2, 3, n = 4, k \in \mathbb{N}^*$$

where  $n$  is the number of the unknown variables.

$$u_1^{(k)} = -6u_3^{(k-1)} + 4u_4^{(k-1)}$$

$$u_2^{(k)} = 4u_3^{(k-1)} - 6u_4^{(k-1)}$$

$$u_3^{(k)} = -6u_1^{(k-1)} + 4u_2^{(k-1)} - \frac{1}{3}$$

$$u_4^{(k)} = 4u_1^{(k-1)} - 6u_2^{(k-1)} + 2$$

Consider the initial solution is

$$u^{(0)} = \left( u_1^{(0)}, u_2^{(0)}, u_3^{(0)}, u_4^{(0)} \right)^T = (0, 0, 0, 0)^T$$

For  $k = 1$  (the first iteration):

$$u_1^{(1)} = -6u_3^{(1-1)} + 4u_4^{(1-1)} = -6u_3^{(0)} + 4u_4^{(0)} = 0$$

$$u_2^{(1)} = 4u_3^{(1-1)} - 6u_4^{(1-1)} = 4u_3^{(0)} - 6u_4^{(0)} = 0$$

$$u_3^{(1)} = -6u_1^{(1-1)} + 4u_2^{(1-1)} - \frac{1}{3} = -6u_1^{(0)} + 4u_2^{(0)} - \frac{1}{3} = -\frac{1}{3}$$

$$u_4^{(1)} = 4u_1^{(1-1)} - 6u_2^{(1-1)} + 2 = 4u_1^{(0)} - 6u_2^{(0)} + 2 = 2$$

So the first approximation is

$$u^{(1)} = \left(u_1^{(1)}, u_2^{(1)}, u_3^{(1)}, u_4^{(1)}\right)^T = \left(0, 0, -\frac{1}{3}, 2\right)^T$$

In similar manner, we can find  $u^{(k)}$  approximation if we know  $u^{(k-1)}$  approximation for any  $k \geq 1$  where  $k \in \mathbb{N}^*$ . Continuing this procedure, we obtain a sequence of approximations.

The following approximate solution  $u$  obtained by Matlab program for Jacobi iterative method with tolerance  $1 \times 10^{-7}$ :

$$u = \begin{bmatrix} 0.222222222222222 \\ 0.444444436729211 \\ 0.111111188263445 \\ 0.222222222222222 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
80	0.005846	$9.258280012081066e - 008$

To see Matlab code for the Jacobi iterative method refer to appendix A.

### **Gauss-Seidel method**

It is given by the sequence (2.4):

$$u_i^{(k)} = \frac{1}{a_{ii}} \left[ -\sum_{j=1}^{i-1} a_{ij} u_j^{(k)} - \sum_{j=i+1}^n a_{ij} u_j^{(k-1)} + b_i \right], a_{ii} \neq 0,$$

$$\text{for } i = 1, 2, \dots, n = 4, k \in \mathbb{N}^*$$

where  $n$  is the number of the unknown variables.

$$u_1^{(k)} = -6u_3^{(k-1)} + 4u_4^{(k-1)}$$

$$u_2^{(k)} = 4u_3^{(k-1)} - 6u_4^{(k-1)}$$

$$u_3^{(k)} = -6u_1^{(k)} + 4u_2^{(k)} - \frac{1}{3}$$

$$u_4^{(k)} = 4u_1^{(k)} - 6u_2^{(k)} + 2$$

Consider the initial solution is

$$u^{(0)} = \left(u_1^{(0)}, u_2^{(0)}, u_3^{(0)}, u_4^{(0)}\right)^T = (0, 0, 0, 0)^T$$

For  $k = 1$  (the first iteration):

$$u_1^{(1)} = -6u_3^{(1-1)} + 4u_4^{(1-1)} = 0$$

$$u_2^{(1)} = 4u_3^{(1-1)} - 6u_4^{(1-1)} = 0$$

$$-6u_1^{(1)} + 4u_2^{(1)} - \frac{1}{3} = -6 \times 0 + 4 \times 0 - \frac{1}{3} = \frac{-1}{3}$$

$$u_3^{(1)} =$$

$$u_4^{(1)} = 4u_1^{(1)} - 6u_2^{(1)} + 2 = 4 \times 0 - 6 \times 0 + 2 = 2$$

So the first approximation is

$$u^{(1)} = \left(u_1^{(1)}, u_2^{(1)}, u_3^{(1)}, u_4^{(1)}\right)^T = \left(0, 0, \frac{-1}{3}, 2\right)^T$$

The following approximate solution  $u$  obtained by Matlab program for Gauss-Seidel iterative method with tolerance  $1 \times 10^{-7}$ :

$$u = \begin{bmatrix} 0.222222057126713 \\ 0.4444444306864854 \\ 0.111111248690702 \\ 0.222222336871881 \end{bmatrix}$$

<i>Number of iterations</i>	<i>cpu – time (seconds)</i>	<i>The error</i>
34	0.002021	$7.264202403489684e - 008$

To see Matlab code for the Gauss-Seidel iterative method refer to appendix B.

### SOR Method

The SOR method is given by the sequence (2.5):

$$u_i^{(k)} = (1 - \omega)u_i^{(k-1)} + \frac{\omega}{a_{ii}} \left[ -\sum_{j=1}^{i-1} a_{ij}u_j^{(k)} - \sum_{j=i+1}^n a_{ij}u_j^{(k-1)} + b_i \right],$$

$$a_{ii} \neq 0, i = 1, 2, 3, n = 4.$$

Choose the relaxation factor  $\omega = 1.3$ :

first, write the Gauss-Seidel equations

$$u_1^{(k)} = -6u_3^{(k-1)} + 4u_4^{(k-1)}$$

$$u_2^{(k)} = 4u_3^{(k-1)} - 6u_4^{(k-1)}$$

$$u_3^{(k)} = -6u_1^{(k)} + 4u_2^{(k)} - \frac{1}{3}$$

$$u_4^{(k)} = 4u_1^{(k)} - 6u_2^{(k)} + 2$$

Now, the SOR equations with  $\omega = 1.3$  are:

$$u_1^{(k)} = (1 - 1.3)u_1^{(k-1)} + (1.3)[-6u_3^{(k-1)} + 4u_4^{(k-1)}]$$

$$u_2^{(k)} = (1 - 1.3)u_2^{(k-1)} + (1.3)[4u_3^{(k-1)} - 6u_4^{(k-1)}]$$

$$u_3^{(k)} = (1 - 1.3)u_3^{(k-1)} + (1.3)[-6u_1^{(k)} + 4u_2^{(k)} - \frac{1}{3}]$$

$$u_4^{(k)} = (1 - 1.3)u_4^{(k-1)} + (1.3)[4u_1^{(k)} - 6u_2^{(k)} + 2]$$

Consider the initial solution is

$$u^{(0)} = \left(u_1^{(0)}, u_2^{(0)}, u_3^{(0)}, u_4^{(0)}\right)^T = (0, 0, 0, 0)^T$$

For  $k = 1$  (the first iteration):

$$u_1^{(1)} = (1 - 1.3)u_1^{(1-1)} + (1.3)[-6u_3^{(1-1)} + 4u_4^{(1-1)}]$$

$$= (-0.3)u_1^{(0)} + (1.3)[-6u_3^{(0)} + 4u_4^{(0)}] = 0$$

$$u_2^{(1)} = (1 - 1.3)u_2^{(1-1)} + (1.3)[4u_3^{(1-1)} - 6u_4^{(1-1)}]$$

$$= (-0.3)u_2^{(0)} + (1.3)[4u_3^{(0)} - 6u_4^{(0)}] = 0$$

$$u_3^{(1)} = (1 - 1.3)u_3^{(1-1)} + (1.3)[-6u_1^{(1)} + 4u_2^{(1)} - \frac{1}{3}]$$

$$= (-0.3)u_3^{(0)} + (1.3)\left[0 + 0 - \frac{1}{3}\right] = -0.43333$$

$$u_4^{(1)} = (1 - 1.3)u_4^{(1-1)} + (1.3)[4u_1^{(1)} - 6u_2^{(1)} + 2]$$

$$= (-0.3)u_4^{(0)} + (1.3)[4 \times 0 - 6 \times 0 + 2] = 2.6$$

So the first approximation is

$$u^{(1)} = \left(u_1^{(1)}, u_2^{(1)}, u_3^{(1)}, u_4^{(1)}\right)^T = (0, 0, -0.43333, 2.6)^T$$

The following approximate solution  $u$  obtained by Matlab program for

SOR iterative method with tolerance  $1 \times 10^{-7}$ :

$$u = \begin{bmatrix} 0.222222201720116 \\ 0.4444444438524804 \\ 0.1111111122578101 \\ 0.222222230932695 \end{bmatrix}$$

<i>Number of iterations</i>	<i>cpu – time (seconds)</i>	<i>The error</i>
15	0.001800	$4.326104671714681e - 008$

To see Matlab code for the SOR iterative method refer to appendix C.

## Conjugate Gradient method

This algorithm can be processed as follows:

**Step 1:** Start with initial guess  $\mathbf{u}_0$ , say

$$\mathbf{u}_0 = (0 \ 0 \ 0 \ 0)^T$$

**Step 2:** Calculate the residual vector  $\mathbf{r}_0$  as follows

$$\begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{u}_0 \\ &= \begin{bmatrix} -1/3 \\ 2 \\ 0 \\ 4/3 \end{bmatrix} - \begin{bmatrix} 6 & -4 & 1 & 0 \\ -4 & 6 & 0 & 1 \\ 1 & 0 & 6 & -4 \\ 0 & 1 & -4 & 6 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

$$\text{So } \mathbf{r}_0 = (-\frac{1}{3}, 2, 0, \frac{4}{3})^T$$

**Step 3:** Let the initial direction vector  $\mathbf{p}_0 = \mathbf{r}_0$ . So

$$\mathbf{p}_0 = (-\frac{1}{3}, 2, 0, \frac{4}{3})^T$$

**Step 4:** Compute the scalars  $\alpha_k$ 's by the formula

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T A \mathbf{p}_k}, \forall k = 0, 1, 2, \dots, n-1$$

For  $k = 0$ :

$$\alpha_0 = \frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{p}_0^T A \mathbf{p}_0}$$

$$\mathbf{r}_0^T \mathbf{r}_0 = \begin{bmatrix} -\frac{1}{3} & 2 & 0 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} -\frac{1}{3} \\ 2 \\ 0 \\ \frac{4}{3} \end{bmatrix} = 5.88889$$

$$\mathbf{p}_0^T A \mathbf{p}_0 = \begin{bmatrix} -1 & 2 & 0 & 4 \\ 3 & & & 3 \end{bmatrix} \begin{bmatrix} 6 & -4 & 1 & 0 \\ -4 & 6 & 0 & 1 \\ 1 & 0 & 6 & -4 \\ 0 & 1 & -4 & 6 \end{bmatrix} \begin{bmatrix} -1 \\ 3 \\ 2 \\ 0 \\ 4 \\ 3 \end{bmatrix} = 46$$

So

$$\alpha_0 = \frac{5.88889}{46} = 0.128019$$

**Step 5:** Compute the first iteration  $\mathbf{u}_1$  by the formula

$$\mathbf{u}_1 = \mathbf{u}_0 + \alpha_0 \mathbf{p}_0$$

$$\begin{aligned} \mathbf{u}_1 &= (0 \ 0 \ 0 \ 0)^T + 0.128019 \left(-\frac{1}{3}, 2, 0, \frac{4}{3}\right)^T \\ &= (-0.042673, 0.256038, 0, 0.170692)^T \end{aligned}$$

The approximate solution  $u$  with tolerance  $1 \times 10^{-7}$  given by Matlab code for conjugate gradient iterative method:

$$u = \begin{bmatrix} 0.222222222222222 \\ 0.444444444444444 \\ 0.111111111111111 \\ 0.222222222222222 \end{bmatrix}$$

<i>Number of iterations</i>	<i>cpu – time (seconds)</i>	<i>The error</i>
4	0.000267	$1.387778780781446e - 016$

To see Matlab code for the conjugate gradient iterative method refer to appendix D.

### Comparison between the iterative methods for example 3.1:

Table 3.1 shows the accuracy for the different iterative methods. That's to say, which of each of the following methods reduce the error

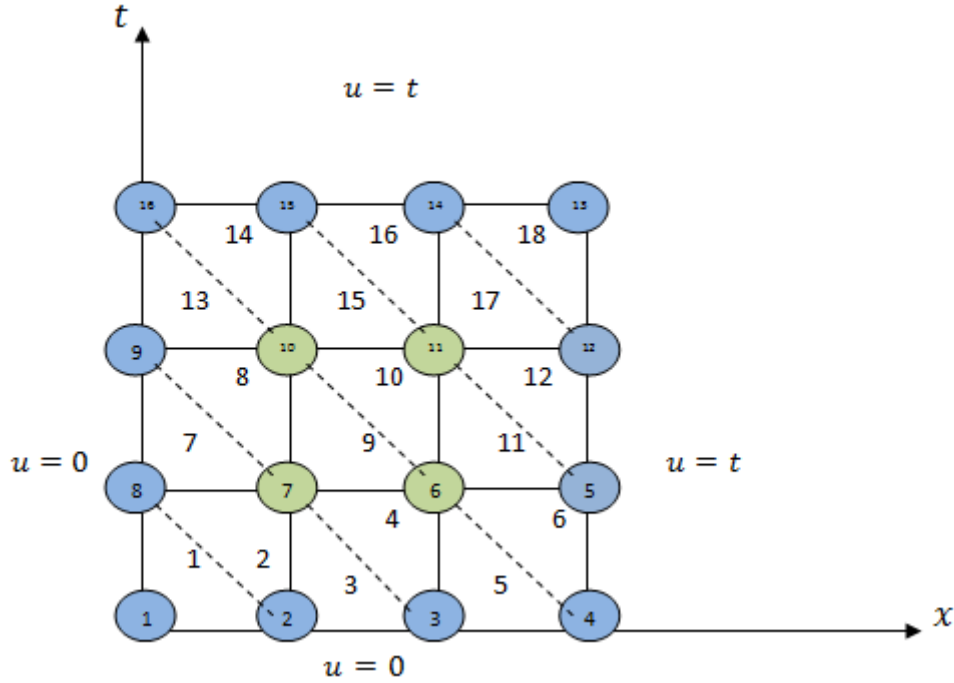


(nearest to the exact solution). Each of them obtains the accurate solution in different number of iterations. However, more iterations give less errors and leads to accurate solutions and this table obtains the cpu time and the error for each method to know the fastest and best method.

**Table 3. 1: Comparison between the iterative methods for example 1**

Method $u$ 's	Jacobi solution	Gauss-Seidel solution	SOR solution	Conjugate gradiante method
$u_1$	0.2222222222222222	0.222222057126713	0.222222201720116	0.2222222222222222
$u_2$	0.444444367292111	0.444444306864854	0.444444438524804	0.4444444444444444
$u_3$	0.111111188263445	0.111111248690702	0.111111122578101	0.1111111111111111
$u_4$	0.2222222222222222	0.222222336871881	0.222222230932695	0.2222222222222222
Number of iterations	80	34	15	4
Cpu-time (seconds)	0.005846	0.002021	0.001800	0.0002667
Error	9.258280012081 $066e - 008$	7.26420240348 $9684e - 008$	4.326104671714 $681e - 008$	1.387778780781 $446e - 016$
<p>The exact solution is:</p> $u = \begin{bmatrix} 0.2222222222222222 \\ 0.4444444444444444 \\ 0.1111111111111111 \\ 0.2222222222222222 \end{bmatrix}$ <p>Tolerance = <math>1e - 7</math></p>				

2- We will apply the finite element method for example 1 to approximate the solution (as shown in figure 3.3).



**Figure 3. 3:** discretization the domain by finite element method for example 2

The region is divided into 18 equal triangular elements which are identified by encircled numbers 1 through 18 as indicated in figure 3.3. In this discretization there are 16 global nodes.

Now, we will write the coordinates for each node:

node 1 :  $(0, 0)$ , node 2 :  $(\frac{1}{3}, 0)$ , node 3 :  $(\frac{2}{3}, 0)$ , node 4 :  $(1, 0)$   
node 5 :  $(1, \frac{1}{3})$ , node 6 :  $(\frac{2}{3}, \frac{1}{3})$ , node 7 :  $(\frac{1}{3}, \frac{1}{3})$ , node 8 :  $(0, \frac{1}{3})$   
node 9 :  $(0, \frac{2}{3})$ , node 10 :  $(\frac{1}{3}, \frac{2}{3})$ , node 11 :  $(\frac{2}{3}, \frac{2}{3})$ , node 12 :  $(1, \frac{2}{3})$   
node 13 :  $(1, 1)$ , node 14 :  $(\frac{2}{3}, 1)$ , node 15 :  $(\frac{1}{3}, 1)$ , node 16 :  $(0, 1)$

For each element  $e$ , we will label the local node numbers 1, 2, and 3 of element  $e$  in a counterclockwise sense.

Table 3.2 shows that for each element we write its global nodes and their local node numbers and coordinates.

**Table 3. 2: the global nodes, local node numbers and the coordinates for each element**

Element #	global nodes	local node numbers	The coordinates of each global node
Element 1	1	1	$(x_1, t_1) = (0, 0)$
	2	2	$(x_2, t_2) = (\frac{1}{3}, 0)$
	8	3	$(x_3, t_3) = (0, \frac{1}{3})$
Element 2	2	1	$(x_1, t_1) = (\frac{1}{3}, 0)$
	8	2	$(x_2, t_2) = (\frac{1}{3}, \frac{1}{3})$
	7	3	$(x_3, t_3) = (0, \frac{1}{3})$
Element 3	2	1	$(x_1, t_1) = (\frac{1}{3}, 0)$
	3	2	$(x_2, t_2) = (\frac{2}{3}, 0)$
	7	3	$(x_3, t_3) = (\frac{1}{3}, \frac{1}{3})$
	...	....	...
Element 17	11	1	$(x_1, t_1) = (\frac{2}{3}, \frac{2}{3})$
	12	2	$(x_2, t_2) = (1, \frac{2}{3})$
	14	3	$(x_3, t_3) = (\frac{2}{3}, 1)$
Element 18	12	1	$(x_1, t_1) = (1, \frac{2}{3})$
	13	2	$(x_2, t_2) = (1, 1)$
	14	3	$(x_3, t_3) = (\frac{2}{3}, 1)$

Now, for each element  $e_i$ , the following must be computed:

For element 1:

$$P_1 = t_2 - t_3 = -\frac{1}{3} \quad Q_1 = x_3 - x_2 = -\frac{1}{3}$$

$$P_2 = t_3 - t_1 = \frac{1}{3} \quad Q_2 = x_1 - x_3 = 0$$

$$P_3 = t_1 - t_2 = 0 \quad Q_3 = x_2 - x_1 = \frac{1}{3}$$

In similar manner, we compute  $P_i'$ s and  $Q_i'$ s for each remaining elements where  $i = 1, 2, 3$ .

We use equation (1.30) to write the entries of the  $3 \times 3$  element coefficient matrix, let us take element 1 as an example:

$$C_{ij}^{(1)} = \frac{1}{4A} [P_i P_j + Q_i Q_j], \text{ for } i, j = 1, 2, 3, \text{ where}$$

$$A = \frac{1}{2} [P_2 Q_3 - P_3 Q_2] = \frac{1}{2} \left[ \frac{1}{3} \cdot \frac{1}{3} - 0 \cdot \frac{1}{3} \right] = \frac{1}{18}$$

$$C_{11}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_1 P_1 + Q_1 Q_1] = \frac{9}{2} \left[ \frac{-1}{3} \cdot \frac{-1}{3} + \frac{-1}{3} \cdot \frac{-1}{3} \right] = 1$$

$$C_{12}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_1 P_2 + Q_1 Q_2] = \frac{9}{2} \left[ \frac{-1}{3} \cdot \frac{1}{3} + \frac{-1}{3} \cdot 0 \right] = -\frac{1}{2}$$

$$C_{13}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_1 P_3 + Q_1 Q_3] = \frac{9}{2} \left[ \frac{-1}{3} \cdot 0 + \frac{-1}{3} \cdot \frac{1}{3} \right] = -\frac{1}{2}$$

$$C_{21}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_2 P_1 + Q_2 Q_1] = \frac{9}{2} \left[ \frac{1}{3} \cdot \frac{-1}{3} + 0 \cdot \frac{-1}{3} \right] = -\frac{1}{2}$$

$$C_{31}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_3 P_1 + Q_3 Q_1] = \frac{9}{2} \left[ 0 \cdot \frac{-1}{3} + \frac{1}{3} \cdot \frac{-1}{3} \right] = -\frac{1}{2}$$

$$C_{22}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_2 P_2 + Q_2 Q_2] = \frac{9}{2} \left[ \frac{1}{3} \cdot \frac{1}{3} + 0 \cdot 0 \right] = \frac{1}{2}$$

$$C_{23}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_2 P_3 + Q_2 Q_3] = \frac{9}{2} \left[ \frac{1}{3} \cdot 0 + 0 \cdot \frac{1}{3} \right] = 0$$

$$C_{32}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_3 P_2 + Q_3 Q_2] = \frac{9}{2} \left[ 0 \cdot \frac{1}{3} + \frac{1}{3} \cdot 0 \right] = 0$$

$$C_{33}^{(1)} = \frac{1}{4 \cdot \frac{1}{18}} [P_3 P_3 + Q_3 Q_3] = \frac{9}{2} \left[ 0 \cdot 0 + \frac{1}{3} \cdot \frac{1}{3} \right] = \frac{1}{2}$$

Thus, the  $3 \times 3$  element coefficient matrix for element 1 is:

$$C^{(1)} = \begin{bmatrix} C_{11}^{(1)} & C_{12}^{(1)} & C_{13}^{(1)} \\ C_{21}^{(1)} & C_{22}^{(1)} & C_{23}^{(1)} \\ C_{31}^{(1)} & C_{32}^{(1)} & C_{33}^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}$$

In a similar manner, we find the  $3 \times 3$  element coefficient matrix for element 2,3,4,...,18.

$$\begin{aligned}
C^{(2)} &= \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix}, C^{(3)} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, C^{(4)} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \\
C^{(5)} &= \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, C^{(6)} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix}, C^{(7)} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, \\
C^{(8)} &= \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix}, C^{(9)} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, C^{(10)} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \\
C^{(11)} &= \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, C^{(12)} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix}, C^{(13)} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix} \\
C^{(14)} &= \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix}, C^{(15)} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, C^{(16)} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \\
C^{(17)} &= \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}, C^{(18)} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix}.
\end{aligned}$$

The global coefficient matrix  $C$  is then assembled from the element coefficient matrices. Since there are 16 nodes, the global coefficient matrix will be a  $16 \times 16$  matrix. The one diagonal entries can be computed as follows:

Take for example :

$$C_{1,1} = C_{11}^{(1)} = 1$$

$$C_{2,2} = C_{22}^{(1)} + C_{11}^{(2)} + C_{11}^{(3)} = 2$$

$$C_{3,3} = C_{22}^{(3)} + C_{11}^{(4)} + C_{11}^{(5)} = 2$$

$$C_{7,7} = C_{22}^{(2)} + C_{33}^{(3)} + C_{33}^{(4)} + C_{22}^{(7)} + C_{11}^{(8)} + C_{11}^{(9)} = 4$$

$$\vdots$$

$$C_{16,16} = C_{33}^{(13)} + C_{33}^{(14)} = 1$$

For the off-diagonal entries ,for example  $C_{7,10}$ , the global link 7–10 corresponds to local link 1–2 of element 8 and local link 1–3 of element 9 as shown in figure 3.3 and hence

$$C_{7,10} = C_{12}^{(8)} + C_{13}^{(9)} = -1$$

We can compute the value of other off-diagonal entries in the same manner.

$$\begin{bmatrix}
 1 & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 \frac{-1}{2} & 2 & \frac{-1}{2} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & \frac{-1}{2} & 2 & \frac{-1}{2} & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & \frac{-1}{2} & 1 & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \frac{-1}{2} & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{2} & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 \\
 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & \frac{-1}{2} & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & \frac{-1}{2} & 1 & \frac{-1}{2} & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & \frac{-1}{2} & 2 & \frac{-1}{2} & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & \frac{-1}{2} & 2 & \frac{-1}{2} \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{2} & 1
 \end{bmatrix}$$

The global coefficient matrix  $C$ . Green numbers are the entries of matrix  $C_{vn}$  while the blue numbers are the entries of matrix  $C_{vv}$ .

Defining the vector  $u_v$  to be vector of unknowns (interior nodes) and vector  $u_n$  to be vector of prescribed boundary values (nodes that are located on the boundaries) as shown in table 3.3.



**Table 3. 3:** represents vector of prescribed boundary values .

Global node	Boundary conditions	The value of global node
1	$u = 0$ on left and down boundaries	The average of its boundary values $\frac{0+0}{2} = 0$
2	$u = 0$ on down boundary	0
3	$u = 0$ on down boundary	0
4	$u = 0$ on down boundary and $u(1, t) = t$ on right boundary	The average of its boundary values $\frac{0+0}{2} = 0$
5	$u(1, t) = t$ on right boundary	$u(1, \frac{1}{3}) = \frac{1}{3}$
8	$u = 0$ on left boundary	0
9	$u = 0$ on left boundary	0
12	$u(1, t) = t$ on right boundary	$u(1, \frac{2}{3}) = \frac{2}{3}$
13	$u(1, t) = t$ on right boundary and $u(x, 1) = t$ on up boundary	The average of its boundary values $\frac{1+1}{2} = 1$
14	$u(x, 1) = x$ on up boundary	$u(\frac{2}{3}, 1) = \frac{2}{3}$
15	$u(x, 1) = x$ on up boundary	$u(\frac{1}{3}, 1) = \frac{1}{3}$
16	$u(x, 1) = x$ on up boundary and $u = 0$ on left boundary	The average of its boundary values $\frac{0+0}{2} = 0$

So,

$$u_n = \left(0, 0, 0, 0, \frac{1}{3}, 0, 0, \frac{2}{3}, 1, \frac{2}{3}, 0\right)^T$$

Now, defining the matrix  $C_{vv}$  to be the matrix of unknown nodes (interior nodes) and the matrix  $C_{vn}$  to be the matrix of unknown nodes and prescribed boundary values. Both matrices  $C_{vv}$  and  $C_{vn}$  are obtained from global coefficient matrix  $C$ .

**Table 3. 4: matrix  $C_{vv}$  that obtained from global coefficient matrix  $C$** 

$C_{vv}$	6	7	10	11
6	4	-1	0	-1
7	-1	4	-1	0
10	0	-1	4	-1
11	-1	0	-1	4

**Table 3. 5: matrix  $C_{vn}$  that obtained from global coefficient matrix  $C$** 

$C_{vn}$	1	2	3	4	5	8	9	12	13	14	15	16
6	0	0	-1	0	-1	0	0	0	0	0	0	0
7	0	-1	0	0	0	-1	0	0	0	0	0	0
10	0	0	0	0	0	0	-1	0	0	0	-1	0
11	0	0	0	0	0	0	0	-1	0	-1	0	0

Now, the inverse of matrix  $C_{vv}^{-1}$  is

$$C_{vv}^{-1} = \begin{bmatrix} 0.2917 & 0.0833 & 0.0417 & 0.0833 \\ 0.0833 & 0.2917 & 0.0833 & 0.0417 \\ 0.0417 & 0.0833 & 0.2917 & 0.0833 \\ 0.0833 & 0.0417 & 0.0833 & 0.2917 \end{bmatrix}$$

The vector  $u_v$  of unknowns nodes can be found by using:

$$u_v = -C_{vv}^{-1} C_{vn} u_n$$

$$\text{Hence } u_v = \begin{bmatrix} \text{node 6} \\ \text{node 7} \\ \text{node 10} \\ \text{node 11} \end{bmatrix} = \begin{bmatrix} 0.2222 \\ 0.1111 \\ 0.2222 \\ 0.4445 \end{bmatrix}.$$

### Example 3.2

Consider the following inhomogeneous one dimensional wave equation

$$\frac{1}{8} u_{xx} - u_{tt} = xt$$

with square domain  $D = \{(x, t) | a = 0 < x < b = 2,$

$c = 0 < t < d = 2\}$  with Neumann boundary condition

$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial x} = g(t) = t$  given on the left boundary and Dirichlet boundary conditions  $u = 1$  on the remaining boundaries.

We will use the finite difference method to approximate the solution of the wave equation.

The mesh size  $h = \frac{1}{2}$  as shown in figure 3.4.

The actual grid points (green points) will be shifted toward the left until locate on the fake boundary (red line). We want to put the grid points  $(1, j)$  outside the domain towards the left .

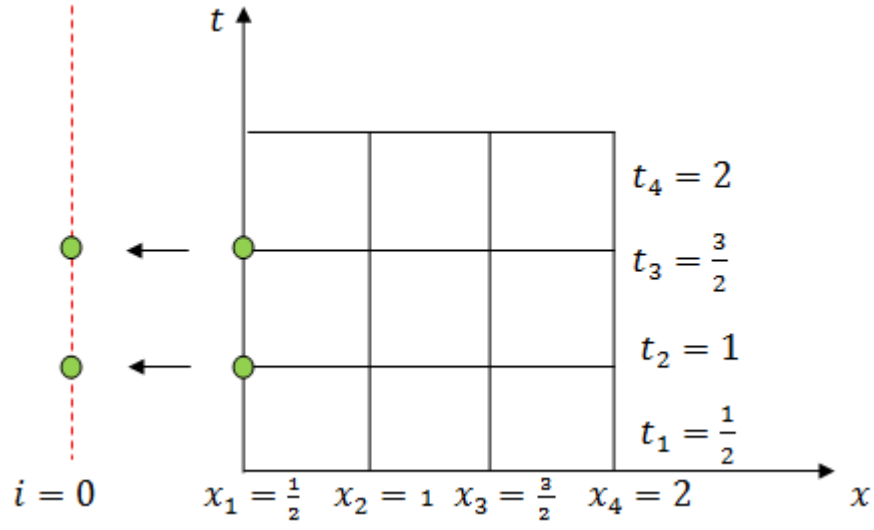
Let  $m = n = 4$ , the following are known as boundary conditions for

$2 \leq i \leq 4 - 1$ :

$$u(2,1) = 1, u(3,1) = 1, u(2,4) = 1, \quad \text{and } u(3,4) = 1$$

And the following are known as boundary conditions for

$$2 \leq j \leq 4 - 1 : u(4,2) = 1, u(4,3) = 1$$



**Figure3. 4:** discretization the domain with Neumann boundary condition for example 3

Now, we use equation (1.29) to approximate the values of boundary points on left boundary :

$$u_{1,j} = \frac{1}{2(1-c^2)} [u_{1,j+1} + u_{1,j-1} - 2c^2 u_{2,j} - 2hc^2 g(1,j) - h^2 G_{1,j}]$$

For  $2 \leq j \leq 4 - 1$ ,  $g_{1,j} = g(x_1, t_j) = g(t_j) = t_j$  and  $G_{i,j} = x_i t_j$

$$\frac{14}{8} u_{1,2} = [u_{1,3} + u_{1,1} - 2\frac{1}{8} u_{2,2} - 2\frac{1}{8} \cdot \frac{1}{2} g(1,2) - \frac{1}{4} x_1 t_2]$$

So

$$\frac{14}{8} u_{1,2} = [u_{1,3} + u_{1,1} - \frac{1}{4} u_{2,2} - \frac{1}{8} - \frac{1}{4} \cdot \frac{1}{2} \cdot 1]$$

But  $u_{1,1}$  is a corner point which we can evaluate its value by equation

(1.21) so,

$$\frac{14}{8} u_{1,2} = u_{1,3} + \frac{1}{2} + \frac{1}{2} u_{1,2} - \frac{1}{4} u_{2,2} - \frac{1}{8} - \frac{1}{4} \cdot \frac{1}{2} \cdot 1$$

Rearrange this equation, then we get

$$\frac{10}{8} u_{1,2} - u_{1,3} + \frac{1}{4} u_{2,2} = \frac{1}{4} \quad (3.7)$$

Now,

$$\frac{14}{8}u_{1,3} = \left[ u_{1,4} + u_{1,2} - 2\frac{1}{8}u_{2,3} - 2\frac{1}{2}\cdot\frac{1}{8}g(1,3) - \frac{1}{4}x_1t_3 \right]$$

$$\frac{14}{8}u_{1,3} = u_{1,4} + u_{1,2} - \frac{1}{4}u_{2,3} - 2\frac{1}{2}\cdot\frac{1}{8}\cdot\frac{3}{2} - \frac{1}{4}\cdot\frac{1}{2}\cdot\frac{3}{2}$$

But  $u_{1,4}$  is a corner point which we can evaluate its value by equation (1.21) so,

$$\frac{14}{8}u_{1,3} = \frac{1}{2} + \frac{1}{2}u_{1,3} + u_{1,2} - \frac{1}{4}u_{2,3} - \frac{1}{8}\cdot\frac{3}{2} - \frac{1}{4}\cdot\frac{1}{2}\cdot\frac{3}{2}$$

Rearrange this equation, then we get

$$\frac{10}{8}u_{1,3} - u_{1,2} + \frac{1}{4}u_{2,3} = \frac{1}{8} \quad (3.8)$$

Now, for  $i = 2,3$  and  $j = 2,3$ , we use equation (1.23)

$$\frac{14}{8}u_{2,2} = u_{2,3} + u_{2,1} - \frac{1}{8}u_{3,2} - \frac{1}{8}u_{1,2} - \frac{1}{4}\cdot 1.1$$

But  $u_{2,1}$  is a known boundary point which is equal to 1 so substitute its value and then rearrange the equation, then we get

$$\frac{14}{8}u_{2,2} - u_{2,3} + \frac{1}{8}u_{3,2} + \frac{1}{8}u_{1,2} = \frac{3}{4} \quad (3.9)$$

$$\frac{14}{8}u_{2,3} = u_{2,4} + u_{2,2} - \frac{1}{8}u_{3,3} - \frac{1}{8}u_{1,3} - \frac{1}{4}\cdot 1\cdot\frac{3}{2}$$

But  $u_{2,4}$  is a known boundary point which is equal to 1 so substitute its value and then rearrange the equation, then we get

$$\frac{14}{8}u_{2,3} - u_{2,2} + \frac{1}{8}u_{3,3} + \frac{1}{8}u_{1,3} = \frac{5}{8} \quad (3.10)$$

$$\frac{14}{8}u_{3,2} = u_{3,3} + u_{3,1} - \frac{1}{8}u_{4,2} - \frac{1}{8}u_{2,2} - \frac{1}{4}\cdot 1\cdot\frac{3}{2}$$

But  $u_{4,2}$  and  $u_{3,1}$  is a known boundary point which are equal to 1 so substitute their value and then rearrange the equation,

Then we get

$$\frac{14}{8}u_{3,2} - u_{3,3} + \frac{1}{8}u_{2,2} = \frac{1}{2} \quad (3.11)$$

$$\frac{14}{8}u_{3,3} = u_{3,4} + u_{3,2} - \frac{1}{8}u_{4,3} - \frac{1}{8}u_{2,3} - \frac{1}{4} \cdot \frac{3}{2} \cdot \frac{3}{2}$$

But  $u_{4,3}$  and  $u_{3,4}$  is a known boundary point which are equal to 1 so substitute their value and then rearrange the equation, then we get

$$\frac{14}{8}u_{3,3} - u_{3,2} + \frac{1}{8}u_{2,3} = -\frac{11}{16} \quad (3.12)$$

Now, we have six equations (3.7, 3.8, 3.9, 3.10, 3.11 and 3.12) in six variables:

$$\begin{aligned} \frac{10}{8}u_{1,2} - u_{1,3} + \frac{1}{4}u_{2,2} &= \frac{1}{4} \\ \frac{10}{8}u_{1,3} - u_{1,2} + \frac{1}{4}u_{2,3} &= \frac{1}{8} \\ \frac{14}{8}u_{2,2} - u_{2,3} + \frac{1}{8}u_{3,2} + \frac{1}{8}u_{1,2} &= \frac{3}{4} \\ \frac{14}{8}u_{2,3} - u_{2,2} + \frac{1}{8}u_{3,3} + \frac{1}{8}u_{1,3} &= \frac{5}{8} \\ \frac{14}{8}u_{3,2} - u_{3,3} + \frac{1}{8}u_{2,2} &= \frac{1}{2} \\ \frac{14}{8}u_{3,3} - u_{3,2} + \frac{1}{8}u_{2,3} &= -\frac{11}{16} \end{aligned}$$

Labeling the variables as follow

$$u_{1,3} = u_1, u_{2,3} = u_2, u_{3,3} = u_3, u_{1,2} = u_4, u_{2,2} = u_5, \text{ and } u_{3,2} = u_6$$

So, the linear system can be written as

$$\begin{aligned} \frac{10}{8}u_4 - u_1 + \frac{1}{4}u_5 &= \frac{1}{4} \\ \frac{10}{8}u_1 - u_4 + \frac{1}{4}u_2 &= \frac{1}{8} \\ \frac{14}{8}u_5 - u_2 + \frac{1}{8}u_6 + \frac{1}{8}u_4 &= \frac{3}{4} \end{aligned}$$

$$\frac{14}{8}u_2 - u_5 + \frac{1}{8}u_3 + \frac{1}{8}u_1 = \frac{5}{8}$$

$$\frac{14}{8}u_6 - u_3 + \frac{1}{8}u_5 = \frac{1}{2}$$

$$\frac{14}{8}u_3 - u_6 + \frac{1}{8}u_2 = -\frac{11}{16}$$

This linear system should be written in matrix form as follows:

$$\begin{bmatrix} \frac{10}{8} & \frac{1}{4} & 0 & -1 & 0 & 0 \\ \frac{1}{8} & \frac{14}{8} & \frac{1}{8} & 0 & -1 & 0 \\ 0 & \frac{1}{8} & \frac{14}{8} & 0 & 0 & -1 \\ -1 & 0 & 0 & \frac{10}{8} & \frac{1}{4} & 0 \\ 0 & -1 & 0 & \frac{1}{8} & \frac{14}{8} & \frac{1}{8} \\ 0 & 0 & -1 & 0 & \frac{1}{8} & \frac{14}{8} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{1}{8} \\ \frac{5}{8} \\ -\frac{11}{16} \\ \frac{1}{4} \\ \frac{3}{4} \\ \frac{1}{2} \end{bmatrix}$$

If we apply Gaussian elimination to this linear system, then we get the following exact solution:

$$u = \begin{bmatrix} -0.285094678094479 \\ 0.996886585608930 \\ -0.508479170819215 \\ -0.232146701215866 \\ 1.020354793701415 \\ -0.077727725732509 \end{bmatrix}$$

We can solve this linear system by any iterative method like Jacobi method, Gauss-Seidel method, Successive over Relaxation (SOR) method and Conjugate Gradient method.

### Jacobi method

The following approximate solution  $u$  obtained by Matlab program for Jacobi iterative method with tolerance  $1 \times 10^{-7}$ :

$$u = \begin{bmatrix} -0.285094424739525 \\ 0.996886506735022 \\ -0.508479153526044 \\ -0.232146409121666 \\ 1.020354725288228 \\ -0.077727705795126 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
95	0.007951	8.867340167695303e-008

To see Matlab code for the Jacobi iterative method refer to appendix E.

### **Gauss-Seidel method**

The following approximate solution  $u$  obtained by Matlab program for Gauss-Seidel iterative method with tolerance  $1 \times 10^{-7}$  :

$$u = \begin{bmatrix} -0.285094420847898 \\ 0.996886526286237 \\ -0.508479158013127 \\ -0.232146481525794 \\ 1.020354743039492 \\ -0.077727714796036 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
49	0.004546	9.547169982360160e – 008

To see Matlab code for the Gauss-Seidel iterative method refer to appendix F.

### **SOR Method**

We recall the SOR iterative method and we choose the relaxation factor  $\omega = 1.3$ :

The following approximate solution obtained by Matlab program for SOR iterative method with tolerance  $1 \times 10^{-7}$  :



$$u = \begin{bmatrix} -0.285094626872160 \\ 0.996886576671534 \\ -0.508479169354012 \\ -0.232146668158265 \\ 1.020354787939089 \\ -0.077727724790212 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
21	0.003395	7.185255812558467e-008

To see Matlab code for the SOR iterative method refer to appendix G .

### **Conjugate Gradient method**

The following approximate solution  $u$  obtained by Matlab program for Conjugate Gradient iterative method with tolerance  $1 \times 10^{-7}$  :

$$u = \begin{bmatrix} -0.285094678094479 \\ 0.996886585608930 \\ -0.508479170819215 \\ -0.232146701215866 \\ 1.020354793701415 \\ -0.077727725732509 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
6	0.000470	1.121023024625157e-016

To see Matlab code for the Conjugate Gradient iterative method refer to appendix H .

### **Comparison between the iterative methods for example 3.2:**

Table 3.2 shows the accuracy for the different iterative methods. That's to say, which of each of the following methods reduce the error (nearest to the exact solution), obtains the accurate solution in different number of iterations, the cpu time and each method error.

**Table 3. 6: Comparison between the iterative methods for example2**

Method $u$ 's	Jacobi solution	Gauss-Seidel solution	SOR solution	Conjugate Gradient solution
$u_1$	-0.285094424739525	-0.285094420847898	-0.285094626872160	-0.285094678094479
$u_2$	0.996886506735022	0.996886526286237	0.996886576671534	0.996886585608930
$u_3$	-0.508479153526044	-0.508479158013127	-0.508479169354012	-0.508479170819215
$u_4$	-0.232146409121666	-0.232146481525794	-0.232146668158265	-0.232146701215866
$u_5$	1.020354725288228	1.020354743039492	1.020354787939089	1.020354793701415
$u_6$	-0.077727705795126	-0.077727714796036	-0.077727724790212	-0.077727725732509
Number of iterations	95	49	21	6
Cpu-time	0.007951	0.004546	0.003395	0.000470
Error	8.867340167695303e-008	9.547169982360160e - 008	7.185255812558467e-008	1.121023024625157e-016
<p>The exact solution is:</p> $u = \begin{bmatrix} -0.285094678094479 \\ 0.996886585608930 \\ -0.508479170819215 \\ -0.232146701215866 \\ 1.020354793701415 \\ -0.077727725732509 \end{bmatrix}$ <p>Tolerance = <math>1e - 7</math></p>				

### Example 3.3

Consider the following two dimensional inhomogeneous wave equation:

$$u_{tt} = 4(u_{xx} + u_{yy}) + xt$$

For  $0 < t < 1$ , the surface  $z = u(x, y, t)$ ,  $0 < x < 1$ ,  $0 < y < 1$ .

Subject to Dirichlet boundary conditions given on the boundaries as follow:  $u(0, y, t) = 0$ ,  $u(1, y, t) = 1$ ,  $u(x, 0, t) = 0$ ,  $u(x, 1, t) = 1$ ,  $u(x, y, 0) = 0$ ,  $u(x, y, 1) = 0$ .

We want to approximate the solution  $u$  by using finite difference method.

Choose positive integers  $m = r = 3$ ,  $n = 2$ .

Define  $\Delta x = \Delta y = h = \frac{b-a}{3} = \frac{1-0}{3} = \frac{1}{3}$ ,  $\Delta t = \frac{1}{3}$ .

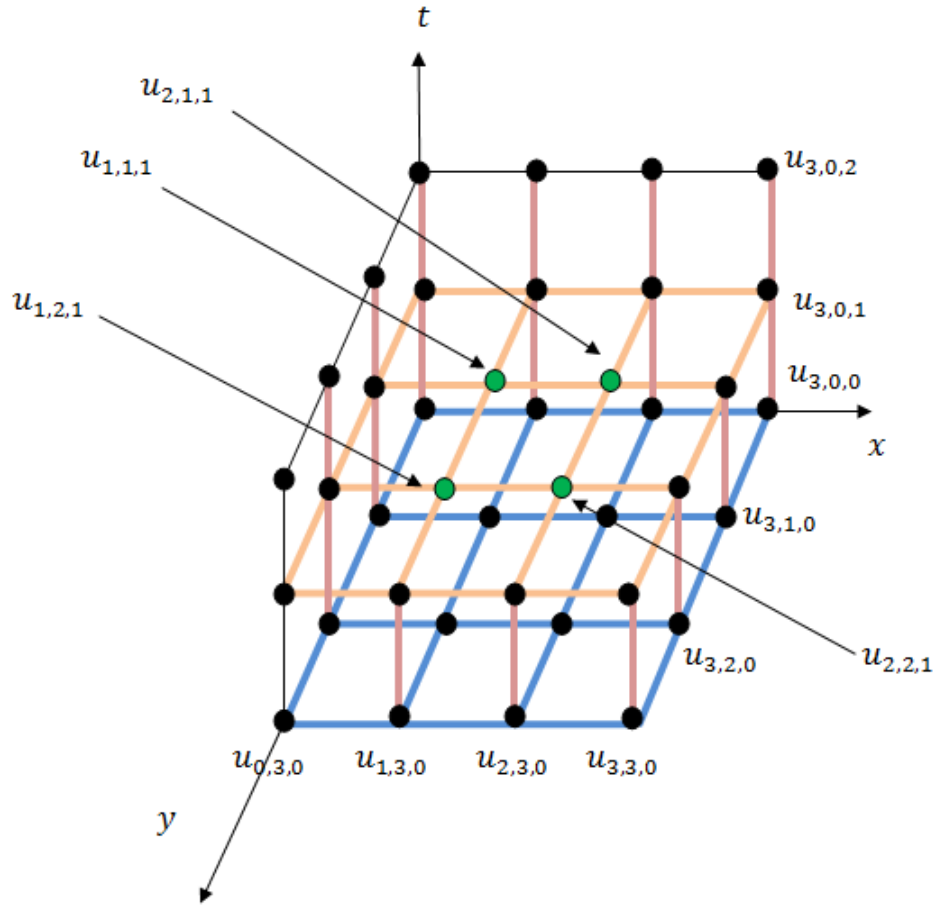
Now, we use the difference equation (1.46) to approximate the interior mesh points (green points shown in figure 3.5) as follows:

$$2(2s^2 - 1)u_{i,j,k} = s^2(u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k}) - u_{i,j,k+1} - u_{i,j,k-1} + (\Delta t)^2 G_{i,j,k}, \text{ for } i = 1, 2, j = 1, 2, k = 1.$$

With boundary conditions:

1.  $u_{0,j,k} = 0$ , for  $j = 0, 1, 2, 3$  and  $k = 0, 1, 2$ .
2.  $u_{m,j,k} = 1$ , for  $j = 0, 1, 2, 3$  and  $k = 0, 1, 2$ .

3.  $u_{i,0,k} = 0$ , for  $i = 1,2$  and  $k = 1$ .
4.  $u_{i,r,k} = 1$ , for  $i = 1,2$  and  $k = 0,1$ .
5.  $u_{i,j,0} = 0$ , for  $i = 1,2,3$  and  $j = 1,2,3$ .
6.  $u_{i,j,n} = 0$ , for  $i = 0,1,2$  and  $j = 0,1,2$ .



**Figure3. 5:** discretization the domain with Dirichlet boundary conditions for example 3

For  $i = 1, j = 1$  and  $k = 1$  :

$$2(2.4 - 1)u_{1,1,1} = 4(u_{2,1,1} + u_{0,1,1} + u_{1,2,1} + u_{1,0,1}) - u_{1,1,2} - u_{1,1,0} + \left(\frac{1}{3}\right)^2 \cdot x_1 \cdot t_1$$

So

$$14u_{1,1,1} = 4(u_{2,1,1} + u_{0,1,1} + u_{1,2,1} + u_{1,0,1}) - u_{1,1,2} - u_{1,1,0} \\ + \left(\frac{1}{3}\right)^2 \cdot \frac{1}{3} \cdot \frac{1}{2}$$

But  $u_{0,1,1}$ ,  $u_{1,0,1}$ ,  $u_{1,1,2}$  and  $u_{1,1,0}$  are boundary points which equal zero's so,

$$14u_{1,1,1} - 4u_{2,1,1} - 4u_{1,2,1} = \frac{-1}{54} \quad (3.13)$$

For  $i = 1, j = 2$  and  $k = 1$  :

$$14u_{1,2,1} = 4(u_{2,2,1} + u_{0,2,1} + u_{1,3,1} + u_{1,1,1}) - u_{1,2,2} - u_{1,2,0} \\ + \left(\frac{1}{3}\right)^2 \cdot \frac{1}{3} \cdot \frac{1}{2}$$

But  $u_{0,2,1} = 0$ ,  $u_{1,3,1} = 1$ ,  $u_{1,2,2} = 0$  and  $u_{1,2,0} = 0$  since it are boundary points so,

$$14u_{1,2,1} - 4u_{2,2,1} - 4u_{1,1,1} = \frac{-217}{54} \quad (3.14)$$

For  $i = 2, j = 1$  and  $k = 1$  :

$$14u_{2,1,1} = 4(u_{3,1,1} + u_{1,1,1} + u_{2,2,1} + u_{2,0,1}) - u_{2,1,2} - u_{2,1,0} \\ + \left(\frac{1}{3}\right)^2 \cdot \frac{2}{3} \cdot \frac{1}{2}$$

But  $u_{3,1,1} = 1$ ,  $u_{2,0,1} = 0$ ,  $u_{2,1,2} = 0$  and  $u_{2,1,0} = 0$  since they are boundary points so,

$$14u_{2,1,1} - 4u_{1,1,1} - 4u_{2,2,1} = \frac{-109}{27} \quad (3.15)$$

For  $i = 2, j = 2$  and  $k = 1$  :

$$14u_{2,2,1} = 4(u_{3,2,1} + u_{1,2,1} + u_{2,3,1} + u_{2,1,1}) - u_{2,2,2} - u_{2,2,0} \\ + \left(\frac{1}{3}\right)^2 \cdot \frac{2}{3} \cdot \frac{1}{2}$$

But  $u_{3,2,1} = 1, u_{2,3,1} = 1, u_{2,2,2} = 0$  and  $u_{2,2,0} = 0$  since they are boundary points so,

$$14u_{2,2,1} - 4u_{1,2,1} - 4u_{2,1,1} = \frac{-217}{27} \quad (3.16)$$

Now, we have four equations (3.13, 3.14, 3.15 and 3.16) in four variables,

$$14u_{1,1,1} - 4u_{2,1,1} - 4u_{1,2,1} = \frac{-1}{54}$$

$$14u_{1,2,1} - 4u_{2,2,1} - 4u_{1,1,1} = \frac{-217}{54}$$

$$14u_{2,1,1} - 4u_{1,1,1} - 4u_{2,2,1} = \frac{-109}{27}$$

$$14u_{2,2,1} - 4u_{1,2,1} - 4u_{2,1,1} = \frac{-217}{27}$$

Labeling the variables as follow

$$u_{1,2,1} = u_1, u_{2,2,1} = u_2, u_{1,1,1} = u_3, \text{ and } u_{2,1,1} = u_4$$

So, the linear system can be written as

$$14u_3 - 4u_4 - 4u_1 = \frac{-1}{54}$$

$$14u_1 - 4u_2 - 4u_3 = \frac{-217}{54}$$

$$14u_4 - 4u_3 - 4u_2 = \frac{-109}{27}$$

$$14u_2 - 4u_1 - 4u_4 = \frac{-217}{27}$$

This linear system could be written in matrix form as

$Au = b$ , where  $u$  is a vector of unknowns

$$A = \begin{bmatrix} 14 & -4 & -4 & 0 \\ -4 & 14 & 0 & -4 \\ -4 & 0 & 14 & -4 \\ 0 & -4 & -4 & 14 \end{bmatrix}, b = \begin{bmatrix} \frac{-217}{27} \\ \frac{54}{27} \\ \frac{-217}{27} \\ \frac{54}{27} \\ \frac{-1}{27} \\ \frac{54}{27} \\ \frac{-109}{27} \end{bmatrix}$$

If we apply Gaussian elimination to this linear system, then we get the following exact solution:

$$u = \begin{bmatrix} -0.670634920634921 \\ -0.957671957671958 \\ -0.384920634920635 \\ -0.671957671957672 \end{bmatrix}$$

We can solve this linear system by any iterative method like Jacobi method, Gauss-Seidel method, Successive over Relaxation (SOR) method or Conjugate Gradient method.

### Jacobi method

The following approximate solution  $u$  obtained by Matlab program for Jacobi iterative method with tolerance  $1 \times 10^{-7}$ :

$$u = \begin{bmatrix} -0.670634815475330 \\ -0.957671852512367 \\ -0.384920529761044 \\ -0.671957566798081 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
28	0.002945	7.886969333181781e – 008

To see Matlab code for the Jacobi iterative method refer to appendix I.

### **Gauss-Seidel method**

The following approximate solution  $u$  obtained by Matlab program for Gauss-Seidel iterative method with tolerance  $1 \times 10^{-7}$  :

$$u = \begin{bmatrix} -0.670634893638285 \\ -0.957671942245309 \\ -0.384920619493986 \\ -0.671957663142444 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
16	0.003186	5.568056038462999e-008

To see Matlab code for the Gauss-Seidel iterative method refer to appendix J.

### **SOR Method**

Choose the relaxation factor  $\omega = 1.3$ :

The following approximate solution  $u$  obtained by Matlab program for SOR method with tolerance  $1 \times 10^{-7}$  :

$$u = \begin{bmatrix} -0.670634937297569 \\ -0.957671955782972 \\ -0.384920624813294 \\ -0.671957667876347 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
15	0.000630	6.288306131363441e-008

To see Matlab code for the SOR iterative method refer to appendix K.



### Conjugate Gradient method

The following approximate solution  $u$  obtained by Matlab program for Conjugate Gradient method with tolerance  $1 \times 10^{-7}$  :

$$u = \begin{bmatrix} -0.670634920634921 \\ -0.957671957671958 \\ -0.384920634920635 \\ -0.671957671957672 \end{bmatrix}$$

<i>Number of iterations</i>	<i>Cpu-time (seconds)</i>	<i>The error</i>
2	0.000215	2.220446049250313e-016

To see Matlab code for the Conjugate Gradient method iterative method refer to appendix L .

**Table 3. 7: Comparison between the iterative methods for example 3**

Method $u$ 's	Jacobi solution	Gauss-Seidel solution	SOR solution	Conjugate Gradient solution
$u_1$	-0.670634815475330	-0.670634893638285	-0.670634937297569	-0.67063492063492121
$u_2$	-0.957671852512367	-0.957671942245309	-0.957671955782972	-0.957671957671958
$u_3$	-0.384920529761044	-0.384920619493986	-0.384920624813294	-0.384920634920635
$u_4$	-0.671957566798081	-0.671957663142444	-0.671957667876347	-0.671957671957672
Number of iterations	28	16	15	2
Cpu-time (seconds)	0.002945	0.003186	0.000630	0.000215
Error	7.8869693331817 81e - 008	5.568056038462999 e-008	6.288306131363441e -008	2.220446049250313e- 016
<p>The exact solution is:</p> $u = \begin{bmatrix} -0.670634920634921 \\ -0.957671957671958 \\ -0.384920634920635 \\ -0.671957671957672 \end{bmatrix}$ <p>Tolerance = <math>1e - 7</math></p>				

### 3.1 Comparison between results for finite difference method and finite element method for example 3.1:

A simple comparison between the results in example 3.1 are shown in table 3.8 and table 3.9 with tolerance  $=1 \times 10^{-7}$ :

**Table 3. 8: Comparison between results for finite difference method and finite element method for example 1**

Finite difference method (using Jacobi method)		Finite element method	
$u_1$	0.2222222222222222	Node 10	0.2222
$u_2$	0.444444367292111	Node 11	0.4445
$u_3$	0.111111188263445	Node 7	0.1111
$u_4$	0.2222222222222222	Node 6	0.2222

**Table 3. 9: Comparison between FE and FD solutions**

$x$	$u(x, \frac{2}{3})$ FE solution	$u(x, \frac{2}{3})$ FD solution
0	0	0
1/3	0.2222	0.2222222222222222
2/3	0.4445	0.444444367292111
1	2/3	2/3

### 3.2 Comparison between results for iterative methods

A simple comparison between the results in example 3.1, example 3.2 and example 3.3 respectively in table 3.1 , table 3.6 and table 3.7 with tolerance  $=1 \times 10^{-7}$ , the comparison yields by compare the number of iterations for converging, error and cpu-time to decide which is the most efficient iterative method .

### 3.3 Conclusions

In this thesis we have used two methods to solve both homogeneous and in homogeneous hyperbolic PDEs subject to Dirichlet and Neumann boundary conditions these methods are the finite difference and the finite element methods. The discretization process transfers the boundary value problem into  $n$  –algebraic linear equations .

This linear system has been solved iteratively by several iterative schemes . These are : Jacobi, Gauss-Seidel, Successive over Relaxation (SOR), and Conjugate Gradient method.

We observe that the finite difference method is very simple and efficient method for approximating the solution of the boundary value problem when the domain has regular shape . While the finite element method is more efficient for irregular domains.

We see clearly that the Conjugate Gradient method is one of the most efficient and accurate method in comparison with the Jacobi, Gauss-Seidel and the SOR methods. In fact, it requires less number of iterations and cpu-time in comparison with the others.

## References

- [1] A. Antunes, R. Leal-Toledo , O. Silveira and E. Toledo, ***Finite Difference Method for Solving Acoustic Wave Equation Using Locally Adjustable Time-steps***, International Conference on Computational Science (ICCS) Journal, Vol. 29, pp. 627–636, 2014.
- [2] W. Bangerth and R. Rannacher, ***Finite Element Approximation of the Acoustic Wave Equation: Error Control and Mesh Adaptation***, East-West Journal of Numerical Mathematics, Volume 7, pp. 263–282, 1999.
- [3] K. Bathe, ***Finite Element Procedures in Engineering Analysis***, Massachusetts Institute of Technology (MIT) Press, 2006.
- [4] V. Bokil and N. Gibson, ***Finite Difference, Finite Element and Finite Volume Methods for the Numerical Solution of PDEs***, Oregon State University Press, DOE Multiscale Summer School, Corvallis, June 30, 2007.
- [5] L. Burden and J. Faires, ***Numerical Analysis*** , (9<sup>th</sup> edition), Brooks Publishing Company, 2011.
- [6] D. Causon and C. Mingham, ***Introductory Finite Difference Methods PDEs***, Ventus Publishing , 2010.
- [7] C. Chua and P. Stoffab, ***Nonuniform Grid Implicit Spatial Finite Difference Method for Acoustic Wave Modeling in Tilted Transversely Isotropic Media***, Journal of Applied Geophysics, Vol. 76, pp. 44– 49 , January 2012.

- [8] R. Clough, *The Finite Element Method in Plane Stress Analysis*, **American Journal of Civil Engineers** , 1960.
- [9] Y. Dong , J. Bancroft and Xiang Du, *2D Wave-Equation Migration by Joint Finite Element Method and Finite Difference Method*, **CREWES Research Journal**, Vol.15, pp. 1-10, 2003.
- [10] R. Glowinski and S. Lapin, *Solution of A wave Equation by A mixed Finite Element — Fictitious Domain Method*, Published by the **Computational Methods in Applied Mathematics Journal**, Vol. 4, pp. 431–444, 2004.
- [11] S. Ham and K. Bathe, *A Finite Element Method Enriched for Wave Propagation Problems*, **Computers and Structures Journal**, Vol. 94-95, pp. 1-12, 2012.
- [12] Y. Hui, *Finite Element Method of the Elastic Wave Equation and Wave-Fields Simulation in Two-Phase Anisotropic Media*, **Chinese Journal of Geophysics**, Vol. 45, pp. 600-610, July 2002 .
- [13] H. Igel, *The Finite Difference Method*, Computational Seismology, Ludwig-Maximilians-University Munich, Oxford University Press, 2002.
- [14] I. Kalambi , *A comparison of Three Iterative Methods for The Solution of Linear Equations*, **Journal of Applied Sciences and Environmental Management (JASEM)**, 2008.
- [15] E. Kreyszig, *Advanced Engineering Mathematics*, 10<sup>th</sup> edition, John Wiley & Sons, New York, 2011.

- [16] M. Lamoureux , ***The Mathematics of PDEs and The Wave Equation*** , University of Calgary, Published by Natural Sciences and Engineering Research Council of Canada (NSERC) Journal, Calgary, August 7–11-2006.
- [17] M. Lamoureux, M. Donald and G. Margrave, ***Galerkin Methods for Numerical Solutions of Acoustic, Elastic and Viscoelastic Wave Equations***, CREWES Research Journal , Vol. 24, 2012.
- [18] J. Lehtinen, ***Time-domain Numerical Solution of the Wave Equation***, Helsinki University of Technology, February 6, 2003.
- [19] K. Lie, ***The Wave Equation in 1D and 2D***, Massachusetts Institute of Technology (MIT) Press, Journal of Computational Physics, University of Oslo ,18.086 Spring 2005.
- [20] L. Lines, R. Slawinski and P. Bording, ***A recipe for Stability Analysis of Finite-Difference Wave Equations Computations***, CREWES Research Journal, Vol.10, pp. 827-830, 1998.
- [21] J. Maloney and M. Kesler ,***"Analysis of Periodic Structures"***  
Chap. 6 in Advances in Computational Electrodynamics: the Finite-Difference Time-Domain Method, A. Taflove, Artech House publishers, Norwood, 1998.
- [22] G. Margrave and F. Mahmoudian, ***A review of The Finite Element Method in Seismic Wave Modelling***, CREWES Research Journal, Vol.15, 2003
- [23] V. Maupin and R. Dmowska , ***The Finite-Difference Time-Domain Method for Modeling of Seismic Wave Propagation***,

Advances in Geophysics, Vol. 48, pp. 421-516, Elsevier - Academic Press, 2007.

- [24] P. Moczo, J. Kristek, M. Galis and P. Pazak, ***On Accuracy of The Finite-Difference and Finite-Element Schemes with Respect to P-Wave to S-Wave Speed Ratio***, Geophysical Journal International, Vol.182, pp. 493–510, July 2010.
- [25] J. Nocedal and S. Wright, ***Numerical Optimization***, Series in Operations Research, Springer Verlag, 1999.
- [26] S. Oliveira, ***A fourth-Order Finite-Difference Method for The Acoustic Wave Equation on Irregular Grids***, Geophysics, Vol. 68, pp. 672–676, 2003.
- [27] L. Olsen-Kettle, ***Numerical Solution of Partial Differential Equations***, University of Queensland, Published by Earth Systems Science Computational Centre, 2011.
- [28] H. Langtangen, ***Finite Difference Methods for Wave Motion***, Numerical Methods for Solving Partial Differential Equations, Computational Partial Differential Equations, second Edition, Springer-Verlag , University of Oslo, 2003.
- [29] W. Press, S. Teukolsky, W. Vetterleng and B. Flannery, ***Numerical Recipes in C***, Second edition, Published by the Press Syndicate of the University of Cambridge , 1992.
- [30] J. Reddy, ***An Introduction to the Finite Element Method***, (Third edition), Vol. 1, McGraw-Hill Series in Mechanical Engineering, New York, 2006.



- [31] Y. Saad, ***Iterative Methods for Sparse Linear Systems***, Second Edition, Society for Industrial and Applied Mathematics (SIAM), 2003.
- [32] J. Saarelma , ***Finite Difference Time-Domain Solver for Room Acoustics Using Graphics Processing Units***, Aalto University School of Electrical Engineering , Master's Thesis Espoo, November 11, 2013.
- [33] M. Sadiku, ***Elements of Electromagnetics***, 4<sup>th</sup> edition, Oxford University Press, New York, 2006.
- [34] R. Sekhar, ***Analytical and Numerical Methods for Ordinary and Partial Differential Equations Arising in Physical Models***, Applied Mathematics, Indian Institute of Technology, Hindawi Publishing Corporation, 2013.
- [35] G. Smith, ***Numerical Solution of Partial Differential Equations***, Finite Difference Methods, Clarendon Press, Oxford University Press, New York , 1987.
- [36] J. Strikwerda, ***Finite Difference Schemes and Partial Differential Equations***, Second Edition, Society for Industrial and Applied Mathematics(SIAM), Philadelphia, 2004.
- [37] J. Thomas, ***Numerical Partial Differential Equations***, Finite Difference Methods, Springer-Verlag, New York, 1995.
- [38] V. Thomee, ***From Finite Differences to Finite Elements, A Short History of Numerical Analysis of Partial Differential Equations***, Elsevier Science B.V., 2001.

- [39] C. Vuik, ***Iterative Solution Methods***, Mathematics and Computer Science, Delft Institute of Applied Mathematics, Morgan Kaufmann Publishers, San Francisco, 2001.
- [40] O. Zienkiewicz and R. Taylor, ***The Finite Element Method : Its Basis and Fundamentals***, Fifth Edition, Vol. 1, Published by Butterworth-Heinemann, Oxford, 2000.

## Appendix A

```
% Matlab code for Jacobi iterative method
% Iterative Solutions of linear equations: Jacobi Method
% Linear system:  $A u = b$ 
% Coefficient matrix A, right-hand side vector b, unknown vector u .

clc
clear

format long

tic

A=[6 -4 1 0;-4 6 0 1;1 0 6 -4;0 1 -4 6];
b=[-1/3;2;0;4/3];

%show the exact solution

inv(A)*b

% Set initial value of u to zero column vector
u=[0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

err=1.0;

k=0;

while err > 1.0e-7

    for i=1:4

        un(i)=(b(i)-(A(i,:)*u-A(i,i)*u(i)))/A(i,i);
```

```
end

err= max(abs(un'-u));

k=k+1;

M(k,:)=un';

u=un';

end

% show the cpu time

toc

% show the solutions

M

% show the error

err

% show the total iteration number

k
```

## Appendix B

```
% Matlab code for Gauss-Seidel iterative method

% Iterative Solutions of linear equations: Gauss-Seidel Method

% Linear system:  $A u = b$ 

% Coefficient matrix A, right-hand side vector b, unknown vector u.

clc

clear

format long

tic

A=[6 -4 1 0;-4 6 0 1;1 0 6 -4;0 1 -4 6];

b=[-1/3;2;0;4/3];

%show the exact solution

inv(A)*b

% Set initial value of u to zero column vector

u=[0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

err=1.0;

k=0;

while err > 1.0e-7

    u0=u;
```

```
for i=1:4
u(i)=(b(i)-(A(i,:)*u-A(i,i)*u(i)))/A(i,i);
end
un=u';
err= max(abs(un'-u0));
k=k+1;
M(k,:)=[u'];
end
% show the cpu time
toc
% show the solutions
M
% show the error
err
% show the total iteration number
K
```

## Appendix C

```
% Matlab code for SOR iterative method

% Iterative Solutions of linear equations: SOR me Method

% Linear system:  $A u = b$ 

% Coefficient matrix A, right-hand side vector b, unknown vector u.

clear

format long

tic

A=[6 -4 1 0;-4 6 0 1;1 0 6 -4;0 1 -4 6];
b=[-1/3;2;0;4/3];

%show the exact solution

inv(A)*b

% Set initial value of u to zero column vector

u=[0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

w=1.02;

err=1.0;

k=0;

while err > 1.0e-7

    u0=u;

    for i=1:4
```

```

    u(i)=(1-w)*u(i)+(w/A(i,i))*(b(i)-(A(i,:)*u-A(i,i)*u(i)));
end
un=u';
    err= max(abs(un'-u0));
    k=k+1;
M(k,:)=u';
end
% show the cpu time
toc
% show the solutions
M
% show the error
err
% show the total iteration number
k

```



## Appendix D

```

function [u, niter, flag] = solveCG(A, f, s, tol, maxiter)

% SOLVECG  Conjugate Gradients method.

%  Input parameters:

%      A : Symmetric, positive definite NxN matrix
%      f : Right-hand side Nx1 column vector
%      s : Nx1 start vector (the initial guess)
%      tol : relative residual error tolerance for break
%            condition
%      maxiter : Maximum number of iterations to perform

%  Output parameters:

%      u : Nx1 solution vector
%      niter : Number of iterations performed
%      flag : 1 if convergence criteria specified by TOL could
%             not be fulfilled within the specified maximum
%             number of iterations, 0 otherwise (= iteration
%             successful).

tic

A=[6 -4 1 0;-4 6 0 1;1 0 6 -4;0 1 -4 6];
f=[-1/3;2;0;4/3];
err=1.0;

format long

s=[0;0;0;0];

tol=0.0000001;

```

```

maxiter = 6;

u = s;      % Set u_0 to the start vector s

r = f - A*s; % Compute first residuum

p = r;

rho = r'*r;

niter = 0;   % Init counter for number of iterations

flag = 0;    % Init break flag

% Compute norm of right-hand side to take relative residuum as
% break condition.

normf = norm(f);

if normf < eps % if the norm is very close to zero, take the
               % absolute residuum instead as break condition
               % ( norm(r) > tol ), since the relative
               % residuum will not work (division by zero).

warning(['norm(f) is very close to zero, taking absolute residuum' ...
        ' as break condition.']);

normf = 1;

end

while (norm(r)/normf > tol) % Test break condition

    a = A*p;

    alpha = rho/(a'*p);

    u = u + alpha*p;

    r = r - alpha*a;

    rho_new = r'*r;

```

```
p = r + rho_new/rho * p;  
rho = rho_new;  
niter = niter + 1;  
if (niter == maxiter)      % if max. number of iterations  
    flag = 1;              % is reached, break.  
    break  
end  
end  
% show the cpu time  
toc  
u  
err= max(abs(u-o))  
niter
```

## Appendix E

```
%Matlab code for Jacobi iterative method

% Iterative Solutions of linear equations: Jacobi Method

% Linear system:  $A u = b$ 

% Coefficient matrix A, right-hand side vector b, unknown vector u.

clc

clear

format long

tic

A=[10/8 1/4 0 -1 0 0;1/8 14/8 1/8 0 -1 0;0 1/8 14/8 0 0 -1;-1 0 0 10/8
1/4 0;0 -1 0 1/8 14/8 1/8;0 0 -1 0 1/8 14/8];

b=[1/8;5/8;-11/16;1/4;3/4;1/2];

%show the exact solution

inv(A)*b

% Set initial value of u to zero column vector

u=[0;0;0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

err=1.0;

k=0;
```

```
while err > 1.0e-7
    for i=1:6
        un(i)=(b(i)-(A(i,:)*u-A(i,i)*u(i)))/A(i,i);
    end
    err= max(abs(un'-u));
    k=k+1;
    M(k,:)=[un'];
    u=un';
end
% show the cpu time
toc
% show the solutions
M
% show the error
err
% show the total iteration number
k
```

## Appendix F

```
% Matlab code for Gauss-Seidel iterative method

% Iterative Solutions of linear equations: Gauss-Seidel Method

% Linear system:  $A u = b$ 

% Coefficient matrix A, right-hand side vector b, unknown vector u

clc

clear

format long

tic

A=[10/8 1/4 0 -1 0 0;1/8 14/8 1/8 0 -1 0;0 1/8 14/8 0 0 -1;-1 0 0 10/8
1/4 0;0 -1 0 1/8 14/8 1/8;0 0 -1 0 1/8 14/8];

b=[1/8;5/8;-11/16;1/4;3/4;1/2];

%show the exact solution

inv(A)*b

% Set initial value of u to zero column vector

u=[0;0;0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

err=1.0;

k=0;

while err > 1.0e-7

    u0=u;
```

```
for i=1:6
u(i)=(b(i)-(A(i,:)*u-A(i,i)*u(i)))/A(i,i);
end
un=u';
err= max(abs(un'-u0));
k=k+1;
M(k,:)=[u'];
end
% show the cpu time
toc
% show the solutions
M
% show the error
err
% show the total iteration number
k
```

## Appendix G

```
% Matlab code for SOR iterative method

% Iterative Solutions of linear equations: SOR Method

% Linear system:  $A u = b$ 

% Coefficient matrix A, right-hand side vector b, unknown vector u

clc

clear

format long

tic

A=[10/8 1/4 0 -1 0 0;1/8 14/8 1/8 0 -1 0;0 1/8 14/8 0 0 -1;-1 0 0 10/8
1/4 0;0 -1 0 1/8 14/8 1/8;0 0 -1 0 1/8 14/8];

b=[1/8;5/8;-11/16;1/4;3/4;1/2];

%show the exact solution

inv(A)*b;

% Set initial value of u to zero column vector

u=[0;0;0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

w=1.3;

err=1.0;

k=0;

while err > 1.0e-7
```



```

u0=u;
for i=1:6
    u(i)=(1-w)*u(i)+(w/A(i,i))*(b(i)-(A(i,:)*u-A(i,i)*u(i)));
end
un=u';
err= max(abs(un'-u0));
k=k+1;
M(k,:)=[u'];
end
% show the cpu time
toc
% show the solutions
M
% show the error
err
% show the total iteration number
k

```

## Appendix H

```

function [u, niter, flag] = solveCG(A, f, s, tol, maxiter)

% SOLVECG  Conjugate Gradients method.

%  Input parameters:

%      A : Symmetric, positive definite NxN matrix
%      f : Right-hand side Nx1 column vector
%      s : Nx1 start vector (the initial guess)
%      tol : relative residual error tolerance for break
%            condition
%      maxiter : Maximum number of iterations to perform

%  Output parameters:

%      u : Nx1 solution vector
%      niter : Number of iterations performed
%      flag : 1 if convergence criteria specified by TOL could
%             not be fulfilled within the specified maximum
%             number of iterations, 0 otherwise (= iteration
%             successful).

tic

A=[10/8 1/4 0 -1 0 0;1/8 14/8 1/8 0 -1 0;0 1/8 14/8 0 0 -1;-1 0 0 10/8
1/4 0;0 -1 0 1/8 14/8 1/8;0 0 -1 0 1/8 14/8];

f=[1/8;5/8;-11/16;1/4;3/4;1/2];

err=1.0;

format long

s=[0;0;0;0;0;0];

```

```

tol=0.0000001;

maxiter =20;

u = s;    % Set u_0 to the start vector s

r = f - A*s; % Compute first residuum

p = r;

rho = r'*r;

niter = 0; % Init counter for number of iterations

flag = 0; % Init break flag

% Compute norm of right-hand side to take relative residuum as
% break condition.

normf = norm(f);

if normf < eps % if the norm is very close to zero, take the
               % absolute residuum instead as break condition
               % ( norm(r) > tol ), since the relative
               % residuum will not work (division by zero).

warning(['norm(f) is very close to zero, taking absolute residuum' ...
        ' as break condition.']);

normf = 1;

end

while (norm(r)/normf > tol) % Test break condition

    a = A*p;

    alpha = rho/(a'*p);

    u = u + alpha*p;

    r = r - alpha*a;

```

```

rho_new = r'*r;
p = r + rho_new/rho * p;
rho = rho_new;
niter = niter + 1;
if (niter == maxiter)      % if max. number of iterations
    flag = 1;              % is reached, break.
    break
end
end
% show the cpu time
toc
u
err= max(abs(u-o))
niter

```

## Appendix I

```
% Matlab code for Jacobi iterative method of FDM for 2D wave
equation

% Iterative Solutions of linear equations: Jacobi Method

% Linear system:  $A u = b$ 

% Coefficient matrix A, right-hand side vector b, unknown vector u.

clc

clear

format long

tic

A=[14 -4 -4 0;-4 14 0 -4 ;-4 0 14 -4;0 -4 -4 14];
b= [-217/54;-217/27;-1/54;-109/27];

%show the exact solution

inv(A)*b

% Set initial value of u to zero column vector

u=[0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$ 

% Show the M matrix

% loop for iterations

err=1.0;

k=0;

while err > 1.0e-7

    for i=1:4
```

```
un(i)=(b(i)-(A(i,:)*u-A(i,i)*u(i)))/A(i,i);  
    end  
    err= max(abs(un'-u));  
    k=k+1;  
M(k,:)=[un'];  
u=un';  
    end  
toc  
% show the solutions  
M  
% show the error  
err  
% show the total iteration number  
k
```

## Appendix J

%Matlab code for Gauss-Seidel iterative method of FDM for 2D wave equation

% Iterative Solutions of linear equations: Gauss-Seidel Method

% Linear system:  $A u = b$

% Coefficient matrix A, right-hand side vector b, unknown vector u.

clc

clear

format long

tic

A=[14 -4 -4 0;-4 14 0 -4 ;-4 0 14 -4;0 -4 -4 14];

b= [-217/54;-217/27;-1/54;-109/27];

%show the exact solution

inv(A)\*b

% Set initial value of u to zero column vector

u=[0;0;0;0];

% Set the iteration number = k , so initial k equals 0

% Set the stopping criteria such that  $err < 1.0e-7$

% Show the M matrix

% loop for iterations

err=1.0;

k=0;

while err > 1.0e-7

u0=u;

```
for i=1:4
u(i)=(b(i)-(A(i,:)*u-A(i,i)*u(i)))/A(i,i);
end
un=u';
err= max(abs(un'-u0));
k=k+1;
M(k,:)=[u'];
end
toc
% show the solutions
M
% show the error
err
% show the total iteration number
k
```



## Appendix K

```
% Matlab code for SOR iterative method of FDM for 2D wave equation
% Iterative Solutions of linear equations: SOR Method
% Linear system:  $A u = b$ 
% Coefficient matrix A, right-hand side vector b, unknown vector u.

clc
clear

format long

tic

A=[14 -4 -4 0;-4 14 0 -4;-4 0 14 -4;0 -4 -4 14];
b=[-217/54;-217/27;-1/54;-109/27];

%show the exact solution

inv(A)*b;

% Set initial value of u to zero column vector
u=[0;0;0;0];

% Set the iteration number = k , so initial k equals 0
% Set the stopping criteria such that  $err < 1.0e-7$ 
% Show the M matrix
% loop for iterations
w=1.3;
err=1.0;
k=0;
while err > 1.0e-7
    u0=u;
```

```

for i=1:4
    u(i)=(1-w)*u(i)+(w/A(i,i))*(b(i)-(A(i,:)*u-A(i,i)*u(i)));
end
un=u';
err= max(abs(un'-u0));
k=k+1;
M(k,:)=[u'];
end
toc
% show the solutions
M
% show the error
err
% show the total iteration number
k

```

## Appendix L

```

function [u, niter, flag] = solveCG(A, f, s, tol, maxiter)

% SOLVECG  Conjugate Gradients method of FDM for 2D wave
equation.

%  Input parameters:

%      A : Symmetric, positive definite NxN matrix
%      f : Right-hand side Nx1 column vector
%      s : Nx1 start vector (the initial guess)
%      tol : relative residual error tolerance for break
%           condition
%      maxiter : Maximum number of iterations to perform

%  Output parameters:

%      u : Nx1 solution vector
%      niter : Number of iterations performed
%      flag : 1 if convergence criteria specified by TOL could
%            not be fulfilled within the specified maximum
%            number of iterations, 0 otherwise (= iteration
%            successful).

tic

A=[14 -4 -4 0;-4 14 0 -4;-4 0 14 -4;0 -4 -4 14];
f= [-217/54;-217/27;-1/54;-109/27];

err=1.0;

format long

s=[0;0;0;0];

```

```

tol=0.0000001;

maxiter =20;

u = s;    % Set u_0 to the start vector s

r = f - A*s; % Compute first residuum

p = r;

rho = r'*r;

niter = 0; % Init counter for number of iterations

flag = 0; % Init break flag

% Compute norm of right-hand side to take relative residuum as
% break condition.

normf = norm(f);

if normf < eps % if the norm is very close to zero, take the
               % absolute residuum instead as break condition
               % ( norm(r) > tol ), since the relative
               % residuum will not work (division by zero).

warning(['norm(f) is very close to zero, taking absolute residuum' ...
        ' as break condition.']);

normf = 1;

end

while (norm(r)/normf > tol) % Test break condition

    a = A*p;

    alpha = rho/(a'*p);

    u = u + alpha*p;

    r = r - alpha*a;

```

```

rho_new = r'*r;
p = r + rho_new/rho * p;
rho = rho_new;
niter = niter + 1;
if (niter == maxiter)      % if max. number of iterations
    flag = 1;              % is reached, break.
    break
end
end
toc
u
err= max(abs(u-o))
niter

```

جامعة النجاح الوطنية  
كلية الدراسات العليا

# طرق عددية لحل مسائل القطع الزائد

إعداد

أنوار جمال محمد عبد الحق

إشراف

أ.د. ناجي قطناني

قدمت هذه الأطروحة استكمالاً لمتطلبات الحصول على درجة الماجستير في الرياضيات  
المحوسبة بكلية الدراسات العليا في جامعة النجاح الوطنية في نابلس - فلسطين.

2017

ب

طرق عددية لحل مسائل القطع الزائد

إعداد

أنوار جمال محمد عبد الحق

إشراف

أ.د. ناجي قطناني

### الملخص

كثيراً من الظواهر الفيزيائية والطبيعية تظهر على شكل نماذج رياضية وتحديداً تظهر كمعادلات تفاضلية جزئية تصف طبيعة هذه الظواهر. في هذه الرسالة استخدمنا المعادلات التفاضلية الجزئية الخطية الزائدة من الدرجة الثانية بحيث تم التركيز على معادلات الموجة كنموذج لوصف تلك الظواهر.

في الواقع، فإن معظم هذه المسائل من الصعب حلها بالطرق التحليلية. بدلا من ذلك، يمكن أن تحل عددياً باستخدام الأساليب الحسابية.

في هذه الأطروحة، معادلات الموجة المتجانسة وغير المتجانسة مع أنواع مختلفة من الشروط الحدية تم حلها عددياً باستخدام طريقة الفروق المحدودة وطريقة العناصر المحدودة لتقريب حل المعادلات التفاضلية الجزئية الزائدة . وبهذا يتم تحويل المعادلة الى شكل آخر للوصول بالنهاية الى نظام خطي من المعادلات يمكن حله باستخدام طرق تكرارية، مثل :

Jacobi, Gauss-Seidel, SOR, and Conjugate Gradient methods. وعمل مقارنة

بسيطة بينهم.

وجدنا في هذا البحث من خلال ما بينته النتائج العددية أن طريقة الفروق المحدودة هي أكثر كفاءة من طريقة العناصر المحدودة للحصول على حل تقريبي للمعادلة وبأقل خطأ ممكن في حال كون المجال ذو أشكال هندسية منتظمة، وأن طريقة العناصر المحدودة أكثر دقة للمجالات المعقدة وغير المنتظمة. أيضاً، نلاحظ أن تقنية Conjugate Gradient تعطي النتائج الأكثر فعالية من بين الطرق التكرارية الأخرى.