An-Najah National University

Faculty of Graduate Studies

Computational Techniques for Solving Linear Parabolic Partial Differential Equation

By

Sameer Mahmoud Musleh

Supervisor

Prof. Naji Qatanani

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

2019

Computational Techniques for Solving Linear Parabolic Partial Differential Equation

By

Sameer Mahmoud Musleh

This thesis was defended successfully on 24/ 3/2019and approved by:

Defense Committee Members	Signature
1. Prof. Dr. Naji Qatanani /Supervisor	.....
2. Dr. Saed Mallak /External Examiner	.....
3. Dr. Adnan Daraghmeh /Internal Examiner	.....

Dedication

I dedicate my work to all my family members, to my parents, my wife, my sisters and 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 the 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 research and I am thankful and indebted to him for sharing expertise, sincere and valuable guidance and encouragement extended to me.

My thanks also to internal examiner Dr. Adnan Daraghmeh and my external examiner Dr. Saed Mallak.

I take the opportunity to express gratitude to all my family members for their help and support.

I also thank my parents, my wife and my daughter for their encouragement, support and attention.

الاقرار

انا الموقع ادناه مقدم الرسالة التي تحمل عنوان:

Computational Techniques for Solving Linear Parabolic Partial Differential Equation

أقر بأن ما اشتملت عليه هذه الرسالة انما هي نتاج جهدي الخاص، باستثناء ما تمت الاشارة اليه حيثما ورد، وان هذه الرسالة ككل، او اي جزء منها لم يقدم من قبل لنيل اي درجة علمية او بحث علمي او بحثي لدى اي مؤسسة تعليمية او بحثية اخرى.

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 Contents

Computational Techniques for Solving Linear Parabolic Partial Differential Equation

By
Sameer Mahmoud Musleh
Supervisor
Prof. Naji Qatanani

Abstract

Parabolic partial differential equations appear in various fields of science and engineering. These involve heat conduction, particle diffusion and ocean propagation. The most common example of such equation is the heat equation.

Physical problems involving parabolic equations are hard to solve analytically, instead, they can be solved numerically using computational methods.

In this work, initial boundary value problems involving heat diffusion phenomenon will be solved. This will be carried out using the finite difference and finite element methods.

The discretizing approach transforms the initial boundary value problem into a linear system of n algebraic equations. Consequently, we use some iterative techniques such as, the Jacobi Method, the Gauss-Seidel Method, the Successive over relaxation (SOR) Method and the Conjugate Gradient Method to solve the resulted linear system. Some numerical test cases will be solved using the proposed methods.

Numerical results show clearly that the finite difference method is more effective than the finite element method for regular domains. Moreover, the results show that the conjugate gradient method gives the most effective results amongst the other iterative schemes.

Introduction

Partial differential equations play a very important role in science, technology and used to describe a wide variety of time dependent phenomena. These include: heat conduction, particle diffusion, ocean acoustic propagation and pricing of derivatives investment.

At the heart of many engineering and scientific analysis is the solution of differential equations, both ordinary and partial differential equations (PDEs). The solution of the later types of equation can be very challenging, depending on the type of equation, the number of independent variables, the boundary and initial conditions and other factors. A variety of broadly applicable methods have been developed to solve such problems. Among the deterministic methods for solving differential equations, are the finite element method and the finite difference method [7, 5]. These methods appear in certain classification of problems for reasons that are deeply rooted in mathematical foundation of each method. Although trends are slowly changing, the finite element method has been traditionally used for solving problems in solid mechanics. While the finite difference method traditionally has been used to solve problems involving fluid flow and heat transfer problems [7, 31].

The finite difference method is one of the oldest and most popular method for solving partial differential equations. This method is based on the application of Taylor expansion used to approximate the solution of partial differential equations [31, 34].

For time dependent problems, considerable progress in finite difference method occurred during the period that followed the end of the Second World War. When the large scale practical applications became possible with the aid of computers, a major role was played by the work of Van Neumann that was partly mentioned in O. Brien Hyman and Kaplan studies in (1951) [31]. As far as parabolic differential equations are concerned, they were highlighted in the early paper done by John in (1951) that included this theory. For initial boundary value problems, implicit methods were established in this period, for example Crale and Nicolson in (1947). The finite difference theory for general initial boundary value problems and parabolic problems then, had an intense period of development during the 50s and the 60s, when the concept of stability was explored in Lax equivalence theorem and the Kreiss matrix lemma, and further major contributions made by Douglas Lees, Samarskii, Widlund and others [31,5].

On the other hand, the finite element method was understood to be used as an approximation for solving partial differential equations utilizing a vartiational principle and piecewise polynomial approximation. G. Leibnitz (1646-1716) in 1696 was the first author to introduce the finite element method. At the same time L. Euler (1707-1783) introduced the vartiational methods with the approximation approach being essentially the main tool employed for derivation of Euler equation [4].

The finite difference and finite element methods are now two universally approaches use to approximate linear and nonlinear differential equations governing mathematical and engineering problems [5,31]. Finite difference method is simple to formulate and can readily be extended to approximate two- or three-dimensional problems. In addition, it is easy to learn, apply and has the flexibility in dealing with problems involving regular geometry. The finite element method has the flexibility in dealing with problems involving irregular geometry. However, with the evolving of numerical grid generation technique, the finite difference method now possesses the geometrical flexibility of finite element method while maintaining the simplicity of the conventional finite difference technique [23].

In the modern era several researchers and authors deal with parabolic partial differential equations and its applications in several fields. Those researchers are: R. Bueckkine, C. Camacho and G. Fabbri worked in economics fields [6]. While F. Abdelnour, H. Voss and A. Raj worked in the field of neuroscience [1]. Other researchers: E. Ostertagova, O. Ostertag and J. Bocko worked in mechanics field [26]. In addition, N. Qatanani worked in the field of heat equation with non-local radiation terms [27].

This thesis is organized as follows: Chapter 1 contains the basic elements and some preliminaries related to second order partial differential equations. In chapter 2 details of finite difference and finite element methods for homogeneous heat equation, with respect to different types of

boundary conditions are presented. Chapter 3 presents some iterative methods namely: the Jacobi, the Gauss-Seidle, the successive over relaxation (SOR) and the Conjugate Gradient methods used for solving linear systems. These systems resulted upon using the finite difference and finite element methods. Chapter 4 contains some numerical examples, comparisons and results.

Chapter One
Mathematical Preliminaries

In this chapter we present some basic definitions and classification of second order linear partial differential equations

1.1 Second Order Linear Partial Differential Equation and their Classification

A second order linear partial differential equation in two variables (x, y) has the general form:

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G(x, y) \qquad (1.1)$$

where the coefficients A, B, C, D, E, F and G can either be constants or functions of variables x and y.

Equation (1.1) can be classified into three types, depending on the discriminant $B^2 - 4AC$ as follows:

1- Hyperbolic

Equation (1.1) is called hyperbolic if the discriminant is positive (i.e. $B^2 - 4AC > 0$). For example, wave equation.

2- Elliptic

Equation (1.1) is called elliptic if the discriminant is negative (i.e. $B^2 - 4AC < 0$). For example, Laplace's equation.

3- Parabolic

Equation (1.1) is called parabolic if the discriminant is equal zero (i.e. $B^2 - 4AC = 0$). For example, heat equation.

In this thesis, we will investigate the linear parabolic partial differential equation with respect to three boundary conditions. These conditions are:

1- Dirichlet Boundary Conditions

The condition where the value of the unknown function is specified on the boundary, $u = g$ on B , with g being a prescribed continuous function on B (Boundaries).

2- Neumann Boundary Conditions

The normal derivative $\frac{\partial u}{\partial n}$ satisfies the condition, $\frac{\partial u}{\partial n} = g$ on B, where g is prescribed function continuous on B (Boundaries).

The symbol $\frac{\partial u}{\partial n}$ denotes the directional derivatives of u along the outward orthogonal to B.

3- Robin's Boundary Conditions (Mixed Boundary Conditions)

These conditions contain the value of the unknown function and its orthogonal derivatives at the boundary of the domain [11].

1.2 Existence of Solution to Heat Equation

Consider the initial boundary value problem

$$\frac{\partial u}{\partial t} - \Delta u = f(x,t), \quad (x,t) \in Q_T \qquad (1.2)$$

$$u(x,t) = \varphi(x,t), \ \ (x,t) \in \partial_T Q_T \qquad (1.3)$$

where $Q_T = \Omega \times (0, \mathrm{T})$, $\Omega \subseteq \mathbb{R}^n$ is bounded domain and $T > 0$.

Theorem 1:[35]

Let $\partial\Omega \in C^{\infty}$, $0 < \alpha < 1$, $f \in C^{\alpha,\frac{\alpha}{2}}(\bar{Q}_T)$ and $\varphi \in C^{2+\alpha,1+\frac{\alpha}{2}}(\bar{Q}_T)$. Then the first initial boundary problem (1.2), (1.3) admits a unique solution $u \in C^{2+\alpha,1+\frac{\alpha}{2}}(\bar{Q}_T)$.

Theorem 2: [35]

Assume that Ω has the exterior ball property and there exist a sequence $\{\Omega_k\}$ with C^{∞} smooth boundary such that $\bar{\Omega}_k \subset \Omega_{k+1}$ and $\partial\Omega_k$ approximate $\partial\Omega$ uniformly. Let $0 < \alpha < 1$, $f \in C^{\alpha,\frac{\alpha}{2}}(\bar{Q}_T)$ and $\varphi \in C^{2+\alpha,1+\frac{\alpha}{2}}(\bar{Q}_T)$. Then the problem (1.2), (1.3) admits a unique solution $u \in C^{2+\alpha,1+\frac{\alpha}{2}}(\bar{Q}_T) \cap C(\bar{Q}_T)$.

Proof:

Without loss of generality, we assume that $\varphi = 0$, otherwise we consider the equation for $w = u - \varphi$ the approximation problem of (1.2), (1.3). We first prove that the limit of the solution of the approximation problems satisfies equation (1.2) and then apply the barrier function technique to check that u at $\partial_T Q_T$ equal zero. Here, we only point out the construction of the barrier function $w(x,t)$. Let $(x^0, t_0) \in \partial_T Q_T$ then the barrier function $w(x,t)$ should have the following properties:

i) $w(x^0, t_0) = 0$, $w(x,t) > 0$ for all $x \in \bar{Q}_T \backslash \{x^0, t_0\}$

ii) $w \in C^{2,1}(\bar{Q}_T)$, $\frac{\partial w}{\partial t} - \Delta w \geq 1$ in $\bar{Q}_T$

Now for point (x^0, t_0) at the lateral boundary, we choose $w(x,t) = w(x)$ and for the point $(x^0, 0) = t$. Clearly the function defined possesses the above properties.

For more details see [35].

1.3 Uniqueness of Solution for Linear Parabolic Partial Differential Equation

To demonstrate uniqueness of solution for one dimensional heat equation with respect to Dirichlet or Neumann boundary conditions, Consider the following problem [15]:

$u_t = u_{xx}$ in $0 < x < L \; and \; t > 0$

with $u = f(x)$ at $t = 0$ and $0 < x < L$

$u(0,t) = g_0(t)$ and $(L,t) = g_L(t)$, $\forall t > 0$ (Dirichlet)

or $\frac{\partial u(0,t)}{\partial x} = g_0(t)$ and $\frac{\partial u(L,t)}{\partial x} = g_L(t)$, $\forall t > 0$ (Neumann)

Suppose that u_1 and u_2 are two solutions and consider $w = u_1 - u_2$ then w satisfies

$w_t = w_{xx}$ in $0 < x < L \; and \; t > 0$

with $w = 0$ at $t = 0$ and $0 < x < L$

$w(0,t) = 0$ and $w(L,t) = 0$, $\forall t > 0$ (Dirichlet)

or $\frac{\partial w(0,t)}{\partial x} = 0$ and $\frac{\partial w(L,t)}{\partial x} = 0$, $\forall t > 0$ (Neumann)

Consider the function of time

$I(t) = \frac{1}{2}\int_0^L w^2(x,t)dx$ such that $I(0) = 0$ and $I(t) > 0, \forall t$. As $(w^2 \geq 0)$ which represents the energy of the function w.

$\frac{dI}{dt} = \int_0^L w\, w_t dx = \int_0^L w\, w_{xx} dx$ (where $w_t = w_{xx}$)

Integrating $\int_0^L w\, w_{xx} dx$ by parts, yields:

$$\left[w\, \frac{\partial w}{\partial x}\right]_0^L - \int_0^L (w_x)^2 dx = -\int_0^L (w_x)^2 dx \leq 0$$

then $0 \leq I(t) \leq I(0) = 0, \forall t > 0$.

Since $\frac{dI}{dt} \leq 0$. So $I(t) = 0, \forall t > 0$ and $w = 0$, that implies $u_1 = u_2$.

Chapter Two
Numerical Techniques

Finite difference and finite element methods are numerical techniques that will be used to discretize the heat equation with respect to different types of boundary conditions.

2.1 Parabolic Equation in One Space Dimension

A linear parabolic partial differential equation takes the general form,

$$\sigma(x,t)u_t = \frac{\partial}{\partial x}(a(x,t)u_x) + b(x,t)u_x + c(x,t)u \qquad (2.1)$$

which is defined within some prescribed domain R of the (x,t) space as shown in Figure (2.1) with this domain, the functions $\sigma(x,t)$ and $a(x,t)$ are strictly positive and $c(x,t)$ is non-negative.

We focus our attention on the finite difference method to discretize parabolic equation as a simplified form of equation (2.1), that is, the diffusion (heat) equation with constant coefficients ($\sigma(x,t) = 1$ $a(x,t) = \alpha^2$ and $b(x,t) = c(x,t) = 0$) [22], i.e.

$$u_t = \alpha^2 u_{xx}, 0 \le x \le L \; and \; t \ge 0 \qquad (2.2)$$

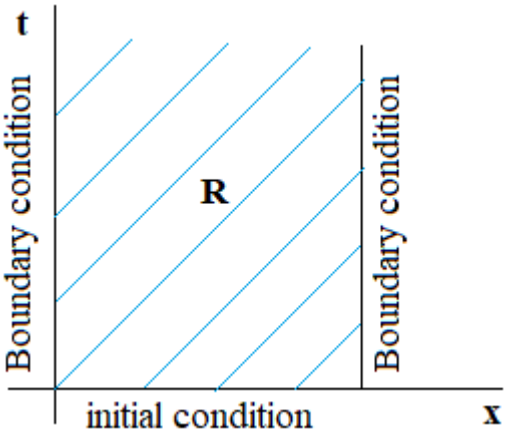

Figure (2.1): Domain of Parabolic equation

2.2 Finite Difference Method Principle

The principle of finite difference method is one of numerical schemes that used to solve partial differential equations. This can be done by replacing the partial derivatives of dependent variables of the unknown function $u(x,t)$ with partial differential equations using finite difference approximation with $O(h^2)$ error. This error is called discretization error or truncation error.

This procedure converts the domain R (where the independent variables are defined) to vertical and horizontal lines called grid lines. Their intersections are called grid points as shown in Figure (2.2).

The replacement of partial derivatives by difference approximation formula depends on Taylor's theorem.

Taylor's theorem [21]

Let $f(x)$ be an $(n+1)$ times differentiable function on an open interval containing a and x. Then

$$f(x) = f(a) + f'(a)\frac{(x-a)}{1!} + f''(a)\frac{(x-a)^2}{2!} + \cdots + f^{(n)}(a)\frac{(x-a)^n}{n!} + E_n(x) \quad (2.3)$$

where $E_n(x) = \frac{(x-a)^{n+1}}{(n+1)!} f^{(n+1)}(c)$

for some number c between a and x

2.3 Strategy of Discretization and Stability Considerations

At this point, finite difference method will be used to discretize the diffusion equation

$$u_t = \alpha^2 u_{xx}, \ 0 \le x \le L \text{ and } t \ge 0$$

The rectangular domain R in Figure (2.1) is converted into identical small rectangles by:

$$x_i = hi \quad \text{and} \quad t_i = kj \quad \text{for} \quad 0 \le i \le n, j \ge 0$$

where

$$h = \frac{\Delta x}{n}, \ k = \frac{\Delta t}{m}$$

as shown in Figure (2.2).

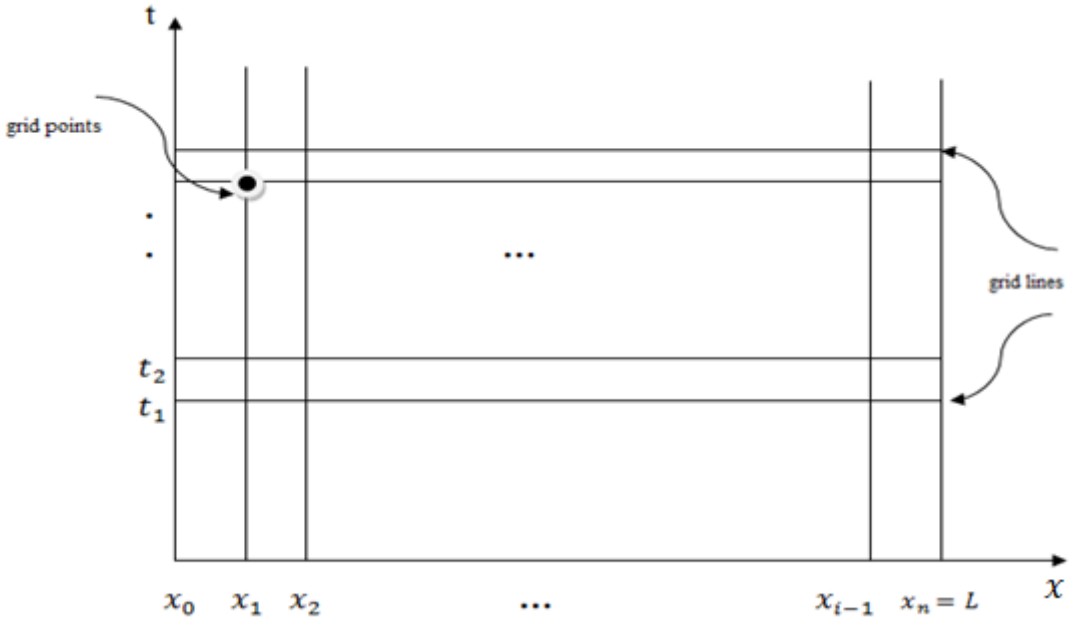

Figure (2.2): Discretize the domain of heat equation

and then replacing partial derivatives u_t and u_{xx} by the value of the unknown function $u(x,t)$ at each grid point i.e. $u(x_i, t_j)$.

To replace partial derivatives of unknown function using Taylor's series, difference method is applied on t, and used on $t_j$and t_{j+1} to obtain

$$\frac{\partial u(x_i,t_j)}{\partial t} = \frac{u(x_i,t_j+k) - u(x_i,t_j)}{k} - \frac{k}{2}\frac{\partial^2 u(x_i,\mu_j)}{\partial t^2} \qquad (2.5)$$

where $\mu_j \in (t_i, t_{j+1})$.

Also, by applying Taylor's theorem in x, we get

$$\frac{\partial^2 u(x_i,t_j)}{\partial x^2} = \frac{u(x_i+h,t_j) - 2u(x_i,t_j) + u(x_i-h,t_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u(\xi_i,t_j)}{\partial x^4} \qquad (2.6)$$

where $\xi_i \in (x_{i-1}, x_{j+1})$.

Substituting (2.5) and (2.6) into (2.2) yields

$$\frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} - \alpha^2 \left[\frac{u(x_i + h, t_j) - 2u(x_i, t_j) + u(x_i - h, t_j)}{h^2}\right] = 0 \quad (2.7)$$

with truncation error

$$\tau_{ij} = \frac{\alpha^2 h^2}{12} \frac{\partial^4 u(\xi_i, t_j)}{\partial x^4} - \frac{k}{2} \frac{\partial^2 u(x_i, \mu_j)}{\partial t^2}$$

for simplicity, we use the notation $u_{i,j}$ to approximate $u(x_i, t_j)$, then equation (2.7) becomes

$$\frac{u_{i,j+1} - u_{i,j}}{k} - \alpha^2 \left[\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}\right] = 0 \quad (2.8)$$

Solving (2.8) for $u_{i,j+1}$, then the finite difference method is called forward-difference method as shown in Figure (2.3). We get,

$$u_{i,j+1} = \left(1 - \frac{2\alpha^2 k}{h^2}\right) u_{i,j} + \frac{\alpha^2 k}{h^2} \left(u_{i+1,j} + u_{i-1,j}\right) \quad (2.9)$$

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

Then, we have

$$u_{0,0} = f(x_0), u_{1,0} = f(x_1), \dots, u_{n,0} = f(x_n)$$

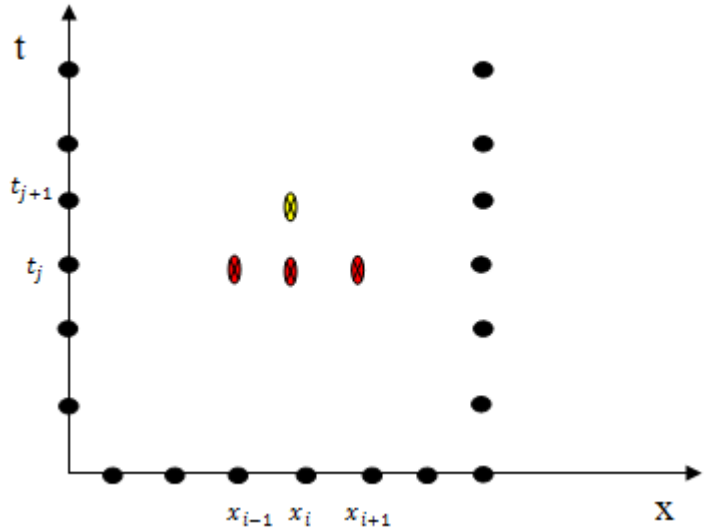

Figure (2.3): Forward-difference method

Then we generate the next t-row by

$$u_{0,1} = u(0, t_1) = 0$$

$$u_{1,1} = \left(1 - \frac{2\alpha^2 k}{h^2}\right) u_{1,0} + \frac{\alpha^2 k}{h^2}\left(u_{2,0} + u_{0,0}\right)$$

$$u_{2,1} = \left(1 - \frac{2\alpha^2 k}{h^2}\right) u_{2,0} + \frac{\alpha^2 k}{h^2}\left(u_{3,0} + u_{1,0}\right) \qquad (2.10)$$

$$u_{n-1,1} = \left(1 - \frac{2\alpha^2 k}{h^2}\right) u_{n-1,0} + \frac{\alpha^2 k}{h^2}\left(u_{n,0} + u_{n-2,0}\right)$$

Now, we can use the $u_{i,1}$ values to generate all $u_{i,2}$ and so on [7,11].

The explicit nature of the difference method implies that the $(n-1) \times (n-1)$ matrix associated with this system (2.10) can be written in tridiagonal form.

$$A=\begin{bmatrix} 1-2\lambda & \lambda & 0 & \cdots & 0 \\ \lambda & 1-2\lambda & \lambda & & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & & & 0 \\ & & & & \lambda \\ 0 & \cdots & 0 & \lambda & 1-2\lambda \end{bmatrix}$$

where $\lambda = \frac{\alpha^2 k}{h^2}$.

So, the approximate solution is demonstrated by $u^{(j)} = Au^{j-1}$, for $j = 1,2,3, \dots$, with error $O(k + h^2)$.

Stability Considerations

Suppose that error $e^{(0)} = (e^{(0)}{}_1, e^{(0)}{}_2, \dots, e^{(0)}{}_{m-1})^t$ is made in representing the initial data

$$u^{(0)} = (f(x_1), f(x_2), \dots, f(x_{m-1}))^t$$

(for any particular step, the choice of the initial step must be convenient). An error of A$e^{(0)}$ propagates in $u^{(1)}$, because

$$u^{(1)} = A\left(u^{(0)} + e^{(0)}\right) = Au^{(0)} + Ae^{(0)}$$

This process continues. At the nth time step, the error in $u^{(n)}$ due to $e^{(0)}$ equals $A^n e^{(0)}$[7]. The forward difference method is consequently stable when these errors do not grow as n increase. But is true if and only if for initial error $e^{(0)}$, we have $\left\|A^n e^{(0)}\right\| \leq \left\|e^{(0)}\right\|$ for any n and any natural

norm (see Definition 3.3). Hence, we must have $\|A^n\| \leq 1$, this condition requires that $\rho(A^n) = (\rho(A))^n < 1$.

The forward difference method is stable if $\rho(A) < 1$.

The eigenvalues of matrix A can be expressed as follow:

$$\propto_i = 1 - 4\,\lambda\,(\sin\left(\frac{i\pi}{2n}\right))^2 \ , \qquad i = 1,2,\dots,n-1$$

so, the condition for stability is reduced to determining whether

$$\rho(A) = max_{1\leq i\leq n-1}\left[1 - 4\,\lambda\,(\sin\left(\frac{i\pi}{2n}\right))^2\right] < 1$$

and this is simplifying to

$0 \leq \lambda\,(\sin\left(\frac{i\pi}{2n}\right))^2 \leq \frac{1}{2}$ for $i = 1,2,\dots,n-1$

Stability requires that this inequality condition holds as $n \to \infty$, the fact that

$$\lim_{n\to\infty}(sin\left(\frac{(n-1)\pi}{2n}\right))^2 = 1$$

that means, stability will occur if $0 \leq \lambda \leq \frac{1}{2}$, by definition λ [?] $= \frac{\alpha^2 k}{h^2}$, so this inequality requires that h and k must be chosen so that,

$$\frac{\alpha^2 k}{h^2} \leq \frac{1}{2} \ .$$

If we use t_j and t_{j-1}, then the finite difference method is called backward finite difference method, as shown in Figure (2.4).

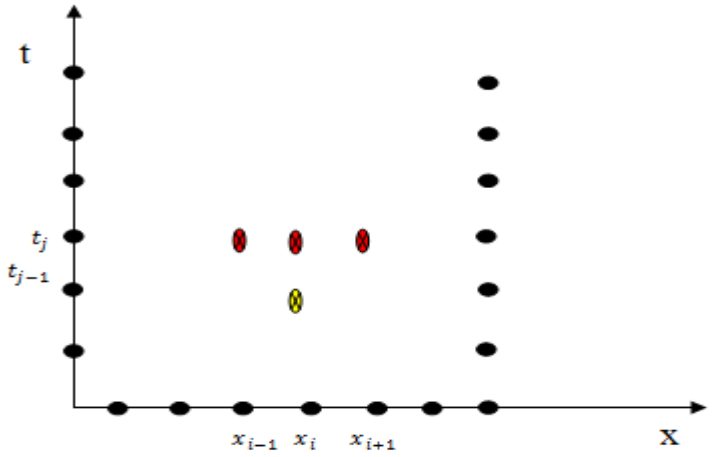

Figure (2.4): Backward-difference method

When considering an implicit difference method that results from using the backward –difference quotient for $u_t(x_i, t_j)$ then it takes the form:

$$\frac{\partial u(x_i,t_j)}{\partial t} = \frac{u(x_i,t_j) - u(x_i,t_{j-1})}{k} - \frac{k}{2}\frac{\partial^2 u(x_i,\mu_j)}{\partial t^2} \qquad (2.11)$$

where $\mu_j \in (t_{i-1}, t_j)$.

Substituting equation (2.5) and (2.6) into (2.11), yields:

$$\frac{u_{i,j} - u_{i,j-1}}{k} - \alpha^2\left[\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}\right] = 0 \qquad (2.12)$$

with truncation error

$$\tau_{ij} = -\frac{\alpha^2 h^2}{12}\frac{\partial^4 u(\xi_i,t_j)}{\partial t^4} - \frac{k}{2}\frac{\partial^2 u(x_i,\mu_j)}{\partial t^2}$$

Solving equation (2.12) for $u_{i,j-1}$, we get:

$$u_{i,j-1} = \left(1 + \frac{2\alpha^2 k}{h^2}\right)u_{i,j} + \frac{\alpha^2 k}{h^2}(u_{i+1,j} + u_{i-1,j}) \qquad (2.13)$$

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

The same argument can be applied on boundaries using the knowledge that $u_{i,0} = f(x_i)$, for $i = 1,2,3,\dots,n-1$. This difference method has the following matrix representation

$$A = \begin{bmatrix} 1+2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\lambda & 1+2\lambda & -\lambda \\ 0 & \cdots & 0 & -\lambda & 1+2\lambda \end{bmatrix}$$

where $\lambda = \frac{\alpha^2 k}{h^2}$

Therefore, the approximate solution is given by $u^{(j-1)} = Au^{(j)}$, for $j = 1,2,\dots$, with truncation error $O(k + h^2)$[7,11].

Stability considerations

The stability for backward-difference method can be illustrated by analyzing the eigenvalues of the matrix A. For the backward-difference method the eigenvalues are:

$$\alpha_i = 1 + 4\,\lambda\,(\sin\left(\frac{i\pi}{2n}\right))^2 \ , \qquad i = 1,2,\dots,n-1$$

since $\lambda \geq 0$, so we have $\alpha_i > 1$, for all $i = 1,2,\dots,n-1$. Since the eigenvalues of A^{-1} are the reciprocals of those of A and the spectral radius $\rho(A^{-1}) < 1$ [7]. This implies that A^{-1} is a convergent matrix.

An error $e^{(0)}$ in the initial data produces an error $(A^{-1})^n e^{(0)}$ at nth step of backward-difference method. Since A^{-1} is convergent, then

$$\lim_{n\to\infty} (A^{-1})^n \, e^{(0)} = 0$$

This means, the method is stable, regardless of the choice of $\lambda = \frac{\alpha^2 k}{h^2}$. This implies that the backward-difference method is unconditionally stable [7].

2.4 Parabolic Partial Differential Equation Subject to Boundary Conditions

Finding the solution for special case of heat equation depending on different types of boundary conditions, namely, Dirichlet, Neumann and Robin's boundary conditions. Consequently, the unknown function must satisfy these conditions at the boundary.

2.4.1 Heat Equation with Dirichlet Boundary Conditions

The Dirichlet boundary conditions, obtained by the German mathematician Dirichlet, is also known as the boundary condition of the first order. In this type of boundary conditions, the value of dependent variable u is prescribed on the boundary.

To derive the formula of finite difference approximation with Dirichlet boundary condition for special case of heat equation

$$u_t = \alpha^2 u_{xx} \qquad (2.14)$$

We consider three points $i-1$, i and $i+1$ which are located on x-axis with an equal distance h between them as shown in Figure (2.5).

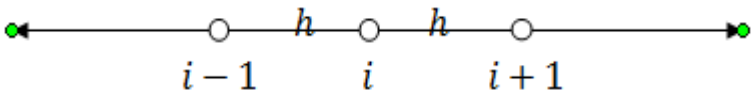

Figure (2.5): Three points $i+1$, i and $i-1$ which are located on x-axis

Consider the points$(i-1,j)$, (i,j) and $(i+1,j)$ are located on t-axis, then the value of unknown function at these points are $u_{i-1,j}$, $u_{i,j}$ and $u_{i+1,j}$ respectively as shown in Figure (2.6)

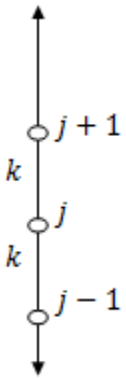

Figure (2.6): Three points $j+1$, j and $j-1$ which are located on t-axis

Now, Taylor's theorem is used to express $u_{i-1,j}$, $u_{i+1,j}$ in the form of Taylor expansion about the point i as follows:

$$u_{i+1,j} = u_{i,j} + h\,\frac{\partial u}{\partial x}|_i + \frac{h^2}{2!}\,\frac{\partial^2 u}{\partial x^2}|_i + \frac{h^3}{3!}\,\frac{\partial^3 u}{\partial x^3}|_i + \frac{h^4}{4!}\,\frac{\partial^4 u}{\partial x^4}|_i + O(h^5) \qquad (2.15)$$

$$u_{i-1,j} = u_{i,j} - h\,\frac{\partial u}{\partial x}|_i + \frac{h^2}{2!}\,\frac{\partial^2 u}{\partial x^2}|_i - \frac{h^3}{3!}\,\frac{\partial^3 u}{\partial x^3}|_i + \frac{h^4}{4!}\,\frac{\partial^4 u}{\partial x^4}|_i + O(h^5) \qquad (2.16)$$

Adding (2.15) and (2.16) yields

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

$$\frac{\partial^2 u}{\partial x^2}|_i = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2) \quad (2.17)$$

Equation (2.17) is a finite difference approximation with error term $O(h^2)$ of second order for $\frac{\partial^2 u}{\partial x^2}|_i$.

Subtracting (2.15) and (2.16) yields

$$\frac{\partial u}{\partial x}|_i = \frac{u_{i+1,j} - u_{i-1,j}}{2h} + O(h^2) \quad (2.18)$$

Similarly, consider three points $j-1, j$ and $j+1$, which are located on t-axis with an equal distance k between them using Taylor expansions to get $u_{i,j-1}$ and $u_{i,j+1}$ about the point j

$$u_{i,j-1} = u_{i,j} - k\,\frac{\partial u}{\partial t}|_j + \frac{k^2}{2!}\,\frac{\partial^2 u}{\partial t^2}|_j - \frac{k^3}{3!}\,\frac{\partial^3 u}{\partial t^3}|_j + \frac{k^4}{4!}\,\frac{\partial^4 u}{\partial t}|_j + \mathrm{O}(h^5) \quad (2.19)$$

$$u_{i,j+1} = u_{i,j} + k\,\frac{\partial u}{\partial t}|_j + \frac{k^2}{2!}\,\frac{\partial^2 u}{\partial t^2}|_j + \frac{k^3}{3!}\,\frac{\partial^3 u}{\partial t^3}|_j + \frac{k^4}{4!}\,\frac{\partial^4 u}{\partial t^4}|_j + \mathrm{O}(h^5) \quad (2.20)$$

Subtracting equations (2.19) and (2.20) yields

$$\frac{\partial u}{\partial t}|_j = \frac{u_{i,j+1} - u_{i,j-1}}{2k} + O(h) \quad (2.21)$$

Substituting equations (2.17) and (2.21) into (2.14), we get

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} = \alpha^2\,\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \mathrm{O}(h^2) \quad (2.22)$$

Solving equation (2.22) for $u_{i,j}$, we get

$$u_{i,j} = \frac{1}{2}\left[u_{i+1,j} + u_{i-1,j}\right] - \frac{h^2}{4k\alpha^2}\left[u_{i,j+1} - u_{i,j-1}\right] \quad (2.23)$$

for $i = 2,3,4,\ldots,n-1$ and $j = 2,3,4,\ldots$

This is valid for any 5 points as shown in Figure (2.7).

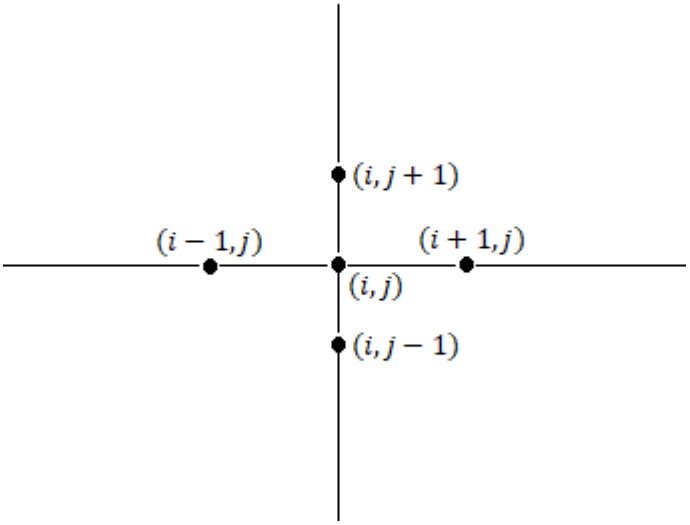

Figure (2.7): Combining x-axis and t-axis around the (i, j) point

Assuming that Dirichlet conditions defined on the semi-rectangular domain $1 \leq i \leq n$ and $j \geq 1$, as shown in Figure (2.8).

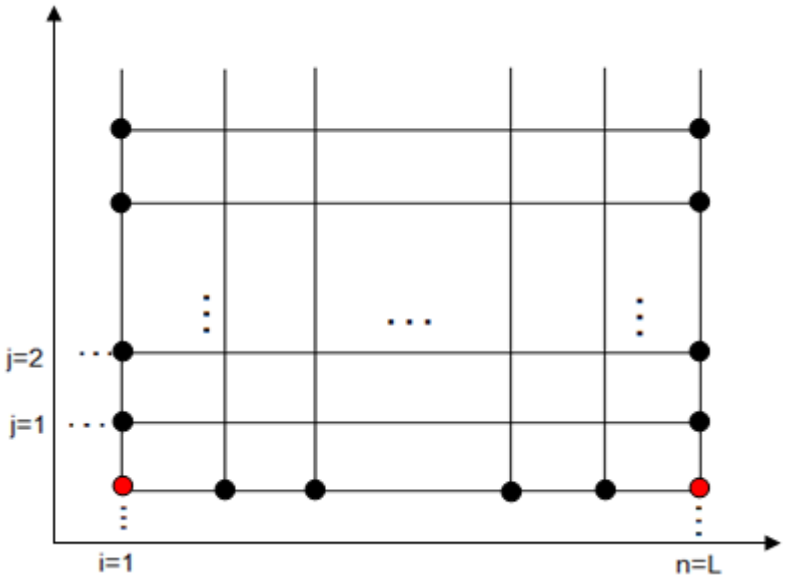

Figure (2.8): Dirichlet boundary conditions defined on the rectangular domain

Let $u(x,t) = g(x,t)$ be given on all boundaries of the domain, so all points in boundary grid (black points) and corner points (red points) are known, and for corner points the following equations are [20, 30]:

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

$$u_{n,1} = \frac{1}{2}u_{n-1,1} - \frac{h}{4}u_{n,2} \qquad (2.24)$$

2.4.2 Heat Equation with Neumann Boundary Conditions

The Neumann boundary conditions, credited to the German mathematician Neumann, is also known as the boundary condition of the second kind. For this type of boundary conditions, the value of the gradient of the dependent variable normal to the boundary $\frac{\partial u}{\partial n}$ is prescribed on the boundary.

The Neumann boundary condition at the left boundary, for example, may be represented as:

$$\frac{\partial u}{\partial n}|_{x=x_0} = \frac{\partial u}{\partial n}|_{i=1} = g(t) \qquad (2.25)$$

where $g(t)$ is the prescribed value of the derivatives.

By applying the second approximation on the approximate equation (2.21) using equation (2.18), the grid points $(1, j)$ are located at imaginary boundary outside the domain towards the left. Here, the grid points fake coordinates become $(0, j)$ as shown in Figure (2.9).

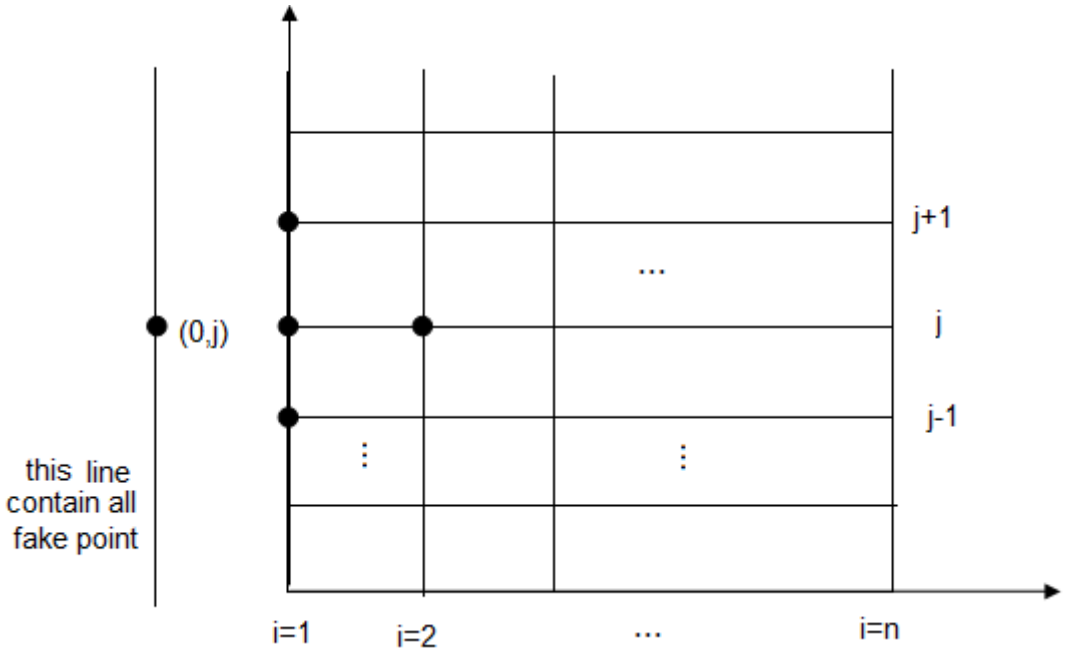

Figure (2.9): Neumann boundary condition defined on the left boundary

Equation (2.23) is approximated using equation (2.16), at the line $i = 1$.

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

thus,

$$u_{0,j} = u_{2,j} + 2kg(1,j) \tag{2.26}$$

Using equation (2.23) at the point $(1,j)$

$$u_{1,j} = \frac{1}{2}\left[u_{2,j} + u_{0,j}\right] - \frac{h^2}{4k}\left[u_{1,j+1} - u_{1,j-1}\right] \tag{2.27}$$

and putting equation (2.26) into (2.27) yields

$$u_{1,j} = \left[u_{2,j} + hg(1,j)\right] - \frac{h^2}{4k}\left[u_{1,j+1} - u_{1,j-1}\right] \tag{2.28}$$

For any two positive integers m and n, we use equation (2.24) for $2 \le i \le n-1$ and $j \ge 2$, where the function $g(1,j)$ is specified [32]. As

Dirichlet condition is specified on the boundary, the value of $\{u_{i+i,j}|2 \leq i \leq n-1\}$,$\{u_{i,j}|2 \leq j\}$, and $\{u_{i,1}|2 \leq i \leq n-1\}$ are known. Equation (2.24) is used to find the corner grid points.

2.4.3 Heat Equation with Robin's Boundary Condition

This type of boundary condition, is a linear combination of the value of the dependent variable and its normal gradient specified at the boundary. This type of boundary condition is credited to the French mathematician Gustave Robin. It is also known as the boundary condition of the third kind, and sometimes referred to as the Robins boundary condition [7]. For one dimensional problem, the Robin boundary condition can be shown as:

$$\tau u(x_L, t_j) + \beta \frac{\partial u}{\partial x}|_{x=x_L} = \gamma \tag{2.29}$$

where τ, β and γ are prescribed constants. To apply the boundary condition, it is first rewritten as follows:

$$\frac{\partial u}{\partial x}|_{i=1} = \frac{\gamma}{\beta} - \frac{\tau}{\beta} u(x_L, t_j) \tag{2.30}$$

substituting equation (2.30) into equation (2.18) yields

$$\frac{\gamma}{\beta} - \frac{\tau}{\beta} u(x_1, t_j) = \frac{u_{1+1,j} - u_{1-1,j}}{2h} \tag{2.31}$$

Rearrangement equation (2.31) we get

$$u_{0,j} = u_{2,j} - 2h\left(\frac{\gamma}{\beta} - \frac{\tau}{\beta} u(x_1, t_j)\right) \tag{2.32}$$

put equation (2.32) into equation (2.8) yields

$$u_{1,j} = \frac{1}{2}\left[2u_{2,j} - 2h\left(\frac{\gamma}{\beta} - \frac{\tau}{\beta}u(x_1, t_j)\right)\right] - \frac{h^2}{4k}\left[u_{1,j+1} - u_{1,j-1}\right]$$

$$u_{1,j} = \left[u_{2,j} - h\left(\frac{\gamma}{\beta} - \frac{\tau}{\beta}u(x_1, t_j)\right)\right] - \frac{h^2}{4k}\left[u_{1,j+1} - u_{1,j-1}\right] \qquad (2.32)$$

and then the same argument for Neumann boundary conditions.

2.5 Finite Element Method

The finite element method is a numerical tool that can be used to determine approximate solution to a large set of partial differential equations.

The finite element method considers the solution region (irregular shape) comprises of many small, interconnected, sub-regions or elements and gives an approximate solution for the governing equations, i.e. the complex partial differential equations are reduced to either linear or nonlinear simultaneous equations. Thus, the finite element discretization procedure reduces the continuum problem, which has finite number of unknown, to one with a finite number of unknowns at specified element points referred to as nodes [3, 4]. Since the finite element method allows us to form the elements, or sub-regions, in arbitrary sense, a close representation of the boundaries of complicated domain is possible.

2.5.1 The Principle of Finite Element Method

The idea behind the finite element method is to divide the solution region into non-over lapping elements or sub-regions.

The finite element allows a variety of element shapes, for example, triangle, rectangle. Each element is formed by connecting a certain number of nodes as shown in Figure (2.10).

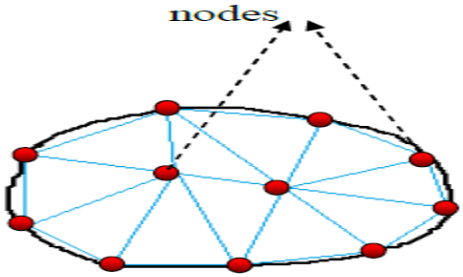

Figure (2.10): Typical finite elements, nodes, edges

2.5.2 Finite Element Method for Dirichlet Boundary Conditions

The finite element method used to discretize the heat equation subject to Dirichlet boundary conditions as follows:

$$u_t = u_{xx},\ 0 \le x \le L, 0 \le t \le T$$

$$u(0,t) = h_1(t),\ u(L,t) = h_2(t)$$

$$u(x,0) = g(x)$$

As illustrated in section (2.3), the solution region is divided into a finite number of elements and triangle elements and the collection of all elements are used to resemble the original region as closely possible as shown in Figure (2.11).

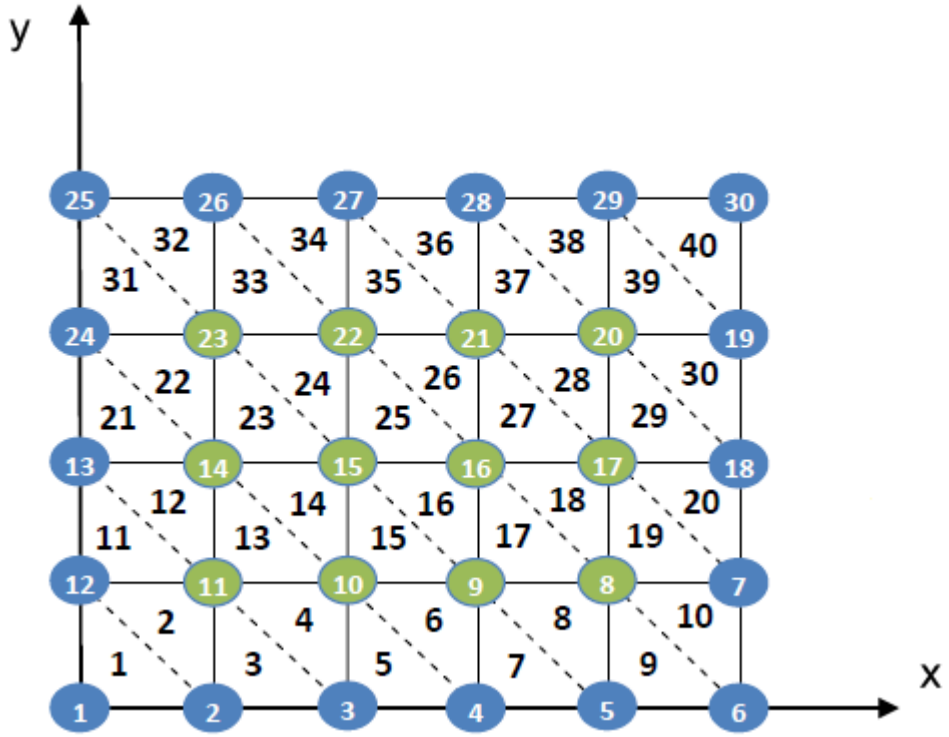

Figure (2.11): Rectangular domain with Dirichlet boundary conditions

For example, the region showed in Figure (2.11), divided into 40 equal triangles. In this discretization, there are thirty global nodes, the blue nodes are known since they are located on the boundaries and interior (green) nodes are unknown.

Assuming that h is the number of equal partitions of $0 \leq x \leq L$ located on the $x - axis$. In our case $h = 5$ (from node 1 to node 2, and from node 2 to node 3 and so on), and the length of each partition is $\frac{L}{h} = \frac{L}{5}$.

Also, let k be the number of equal partitions of $0 \leq t \leq T$ located on t-axis. In our case $k = 4$, (from node 1 to node 12, and node 12 to node 13 and so on), and the length of each partition is $\frac{T}{k} = \frac{T}{4}$.

The coordinate for each node can be determined with respect to partition as shown in Figure (2.12) as follows:

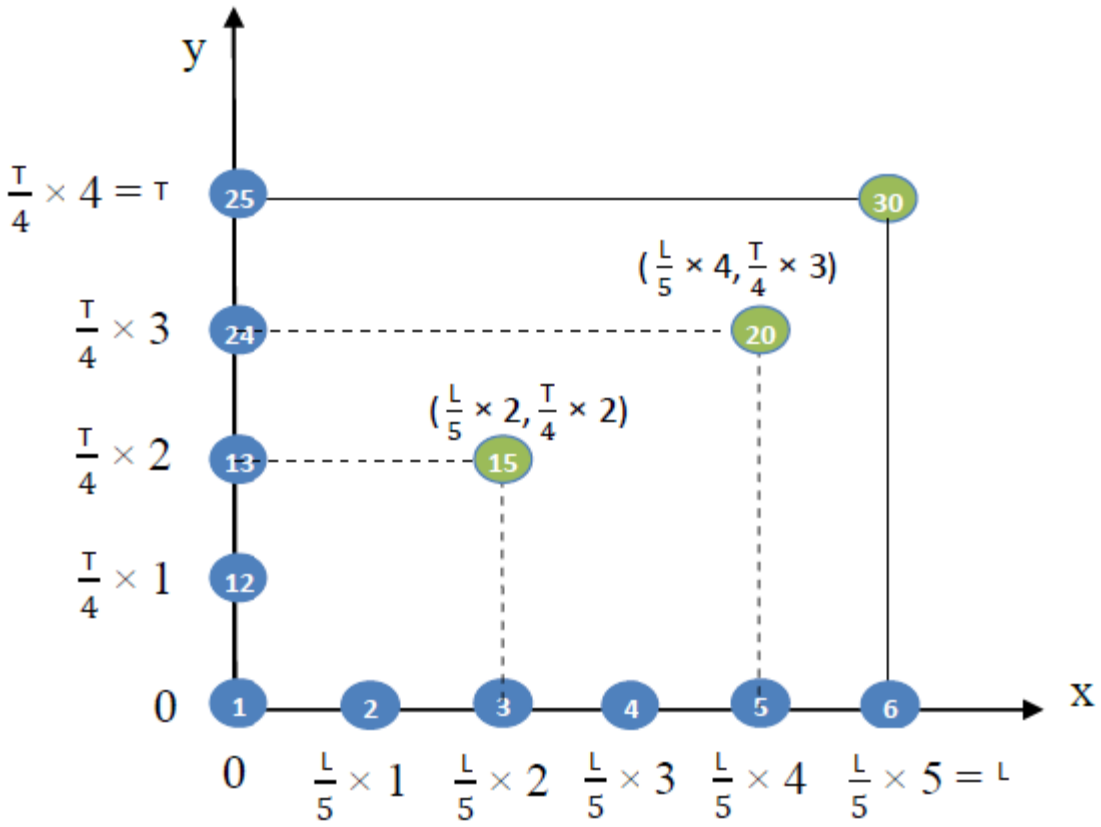

Figure (2.12): Coordinate for each node in finite element method

node 1: $(0, 0)$, node 2: $(\frac{L}{5} \times 1, 0)$, node 3: $(\frac{L}{5} \times 2, 0)$, …, node 30: (L, T).

Now, when deriving governing equation, for a typical element we determine the coefficient matrix. For each element (triangle) i, we have nodes 1, 2 and 3 that must be assigned, so that global nodes associated with an element are traversed in a counter clockwise. If we take element 1 and locate each element coordinate as follows:

the local node 1 is coordinate $(x_1, t_1) = (0,0)$

the local node 2 is coordinate $(x_2, t_2) = (\frac{L}{5}, 0)$

the local node 3 is coordinate $(x_3, t_3) = (0, \frac{T}{4})$

as shown in Figure (2.13).

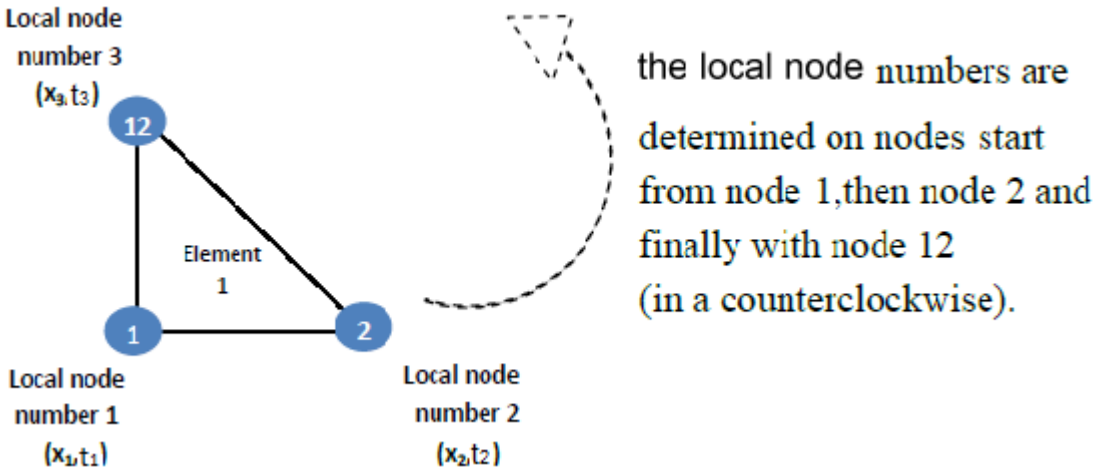

Figure (2.13): The local node numbers are determined on nodes start form node 1, then node 2 and finally with node 12 (in a counterclockwise)

Similarly, the local node is determined for each element in the same way to find coefficient matrix [9, 18].

For each element *e* the following quantities are computed:

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

$$Q_2 = x_1 - x_3 \, , P_2 = t_3 - t_1 \qquad (2.34)$$

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

where the subscripts refer to the local node number 1, 2 and 3 of element i. For example, in Figure (2.11), element (25) has global node 15, 16 and 22 respectively.

For each element P_i and Q_i for $i = 1,2,3$ are computed to obtain 3×3 element coefficient matrix by

$$C_{ij}^{e} = \frac{1}{4A}[P_i P_j + Q_i Q_j],\ i,j = 1,2,3 \qquad (2.35)$$

where

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

The global coefficient matrix is assembled from the element's coefficient matrices. Since there are 30 nodes, the global coefficient matrix will be a 30×30 matrix.

The computation of one diagonal and off-diagonal entries illustrated in the following example, node 13, which corresponds to the $C_{13,13}$ entry in the global coefficient matrix C, belongs to element 11, 12 and 21, since node 13 is assigned local node number 3 in element 11 and 12, and local node number 1 in element 21, as shown in Figure (2.11), the corresponding global coefficient is

$$C_{13,13} = C_{3,3}^{(11)} + C_{3,3}^{(12)} + C_{1,1}^{(21)}$$

For off-diagonal entry $C_{10,15}$, global link 10-15 corresponds to local link 1-2 of element 14 and local link 1-3 of element 15, hence

$$C_{10,15} = C_{1,2}^{(14)} + C_{1,3}^{(15)}$$

Define u_r to be the vector of unknown nodes (interior nodes) and u_s to be the vector of prescribed boundary values as shown in Figure (2.11).

Define matrix C_{rr} to be a matrix of unknown nodes obtained from the global coefficient matrix C and matrix C_{rs} to be a matrix of unknown nodes with prescribed boundary values that are obtained from the global coefficient matrix.

In our case, C_{rr} is a 12×12 matrix since we have 12 interior nodes (green nodes) and C_{rs} is 12×18 matrix since we have 12 interior nodes and 18 boundary nodes (blue nodes). Also, u_s is a vector of size 18×1.

The vector u_r of unknown nodes can be computed by using:

$$u_r = -C_{rr}^{-1} C_{rs} u_s \qquad (2.36)$$

The vector u_r contains the approximation to the unknown nodes (interior nodes) [18, 13].

2.5.3 Finite Element Method for Neumann Boundary Conditions

Consider the one-dimensional heat equation with Neumann boundary condition, that is

$$u_t - u_{xx} = 0, \quad 0 \le x \le l \quad , \quad t \ge 0$$

$$u(x,0) = h(x) \qquad 0 \le x \le l$$

$$u_x(0,t) = g(t) \qquad t \geq 0$$

Weak formulation starts by multiplying the partial differential equation by test function $v(x) \in H^1(\Omega)$ on both sides, then integrate the resulting equation over the domain [37]. We obtain the weak formulation:

$$\int_{I_n} \int_0^l u_t v(x) dx dt = \int_{I_n} \int_0^l u_{xx} v(x) dx dt \tag{2.37}$$

Integrating the right-hand side of equation (2.37) by part, we get

$$\int_{I_n} \int_0^l u_t v(x) dx dt = \int_{I_n} [u_x v(x)]_0^l - \int_{I_n} \int_0^l u_{xx} v(x) dx dt \tag{3.38}$$

Inserting the boundary condition in equation (3.38) yield

$$\int_{I_n} \int_0^l u_t v(x) dx dt$$

$$= \int_{I_n} u_x(l)v(l) - g(t)v(0) - \int_{I_n} \int_0^l u_{xx} v(x) dx dt \tag{3.39}$$

we can express equation (2.37) as:

$$(u_t, v(x)) = -a\big(u, v(x)\big) \qquad \forall v(x) \epsilon H^1(\Omega) \tag{2.40}$$

where $a\big(u, v(x)\big) = \int_{I_n} \int_0^l u_{xx} v(x) dx dt$

Given a triangulation T_n and finite space $v_h \in H^1(\Omega) \cap C^0(\Omega)$, with

$\phi_i(x), i = 1,2,3, \ldots$, denote a set of basis function for v_h to seek the finite element solution of form:

$$u(x,t) = \sum_{j=1}^{M} u_j(t)\phi_j(x) \tag{2.41}$$

Substituting (2.40) into (2.39) to get

$$\left(\sum_{j=1}^{M} u_i'(t)\phi_i(x)\,, v_h\right) = -a\left(\sum_{j=1}^{M} u_i(t)\phi_i(x)\,, v_h\right) \tag{2.42}$$

and let $v_h(x) = \phi_i(x)$ for $i = 1,2,3, \ldots$, to obtain the following linear system of ODEs:

$$\begin{bmatrix} (\phi_1,\phi_1) & \cdots & \cdots & (\phi_1,\phi_M) \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ (\phi_M,\phi_1) & \cdots & \cdots & (\phi_M,\phi_M) \end{bmatrix} \begin{bmatrix} u_1'(t) \\ u_2'(t) \\ \vdots \\ u_M'(t) \end{bmatrix}$$
$$= \begin{bmatrix} a(\phi_1,\phi_1) & \cdots & \cdots & a\ (\phi_1,\phi_M) \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a(\phi_M,\phi_1) & \cdots & \cdots & a(\phi_M,\phi_M) \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_M(t) \end{bmatrix}$$

The corresponding problem can be expressed as:

$$B\frac{du}{dt} + Au(t) = 0 \tag{2.43}$$

with initial condition $u(x,0) = h(x)$, $\quad \forall i = 1,2,3, \ldots$

Using the forward finite difference approximation, we get

$$u_t = \frac{u_{k+1} - u_k}{\Delta t} \tag{2.44}$$

Inserting (2.44) into (2.43) yields

$$B\frac{u_{k+1} - u_k}{\Delta t} + Au_k = 0 \tag{2.45}$$

Solving equation (2.45) with respect to u_{k+1}, we have

$$u_{k+1} = B^{-1}(B - \Delta t A)u_k \tag{2.46}$$

2.5.4 Finite Element Method with Robin's Boundary Conditions

Consider the one-dimensional heat equation with Robin's boundary conditions:

$$\begin{cases} u_t - u_{xx} = 0 \\ \alpha u(l,t) + \beta u_x(0,t) = g(t) \end{cases} \tag{2.46}$$

where α and β are constants.

We do same argument in previous section, by multiplying both sides of equation (2.46) by test function $v(x)$, then integrating by parts and substitute the condition $u_x(0,t) = \frac{g(t)-\alpha u(l,t)}{\beta}$ in equation (2.39) yields:

$$\int_{I_n}\int_{0}^{l} u_t v(x)dxdt = \int_{I_n} u_x(l)v(l) - \frac{g(t)-\alpha u(l,t)}{\beta}v(0) - \int_{I_n}\int_{0}^{l} u_{xx}v(x)dxdt \tag{2.47}$$

The same procure in pervious section (2.5.3) to get the formula of finite element for heat equation with Robin's boundary condition.

Chapter Three

Iterative Methods for Solving Linear Systems

In previous chapter, finite difference and finite element methods are used to discretize the partial differential equations. This discretization yields a system of linear equations which can be solved by different iterative schemes [10]. In this chapter we will use the Jacobi, the Gauss-Seidel, the Successive over Relaxation and the Conjugate Gradient methods to solve this linear system and discuss their convergence properties.

For solving the $n \times n$ linear system

$$A\mathbf{x} = b \tag{3.1}$$

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 solution $\mathbf{x}$.

Most iterative methods involve a process of converting the system $A\mathbf{x} = b$ into an equivalent system:

$$\mathbf{x} = T\mathbf{x} + c \tag{3.2}$$

where T is an $n \times n$ matrix and c is a column matrix.

After selecting an initial approximation $\mathbf{x}^{(0)}$, we generate a sequence of vectors $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ defined as [11]:

$$\mathbf{x}^{(k)} = T\,\mathbf{x}^{(k-1)} + \boldsymbol{c}, \quad k \geq 1 \tag{3.3}$$

Four iterative methods: Jacobi method, Gauss-Seidle method, successive over relaxation (SOR) method and Gradient method, are to be considered.

3.1 Jacobi Method

The Jacobi method is the simplest iterative method for solving a (square) linear system. This method depends on two assumptions: the linear system $A\mathbf{x} = \boldsymbol{b}$ has a unique solution and the coefficient matrix A has no zeros on its main diagonal. If any of the diagonal entries are zero, then rows and columns must be interchanged to get a coefficient matrix that has nonzero entries in the main diagonal [7].

To derive a general formula of Jacobi method, consider the following $n \times n$ linear system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{3.4}$$

We can rewrite (3.4) in a matrix form:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.5}$$

where

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

The equation (3.4) can be expressed in the form $\mathbf{x} = T\mathbf{x} + c$ as follows:

$$\begin{aligned} x_1^{(k)} &= \frac{-a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} - \dots - \frac{a_{1n}}{a_{11}} x_n^{(k-1)} + \frac{b_1}{a_{11}} \\ x_2^{(k)} &= \frac{-a_{21}}{a_{22}} x_1^{(k-1)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} - \dots - \frac{a_{2n}}{a_{22}} x_n^{(k-1)} + \frac{b_2}{a_{22}} \\ &\vdots \\ x_n^{(k)} &= \frac{-a_{n1}}{a_{nn}} x_1^{(k-1)} - \frac{a_{n2}}{a_{nn}} x_2^{(k-1)} - \dots - \frac{a_{nn-1}}{a_{nn}} x_{n-1}^{(k-1)} + \frac{b_n}{a_{nn}} \end{aligned} \tag{3.6}$$

The system (3.6) can be illustrated into matrix form

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 & \frac{-a_{12}}{a_{11}} & \dots & \frac{-a_{1n}}{a_{11}} \\ \frac{-a_{21}}{a_{22}} & 0 & \dots & \frac{-a_{2n}}{a_{22}} \\ \vdots & & \ddots & \vdots \\ \frac{-a_{n1}}{a_{nn}} & \frac{-a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix} \tag{3.7}$$

where

$$T = \begin{bmatrix} 0 & \frac{-a_{12}}{a_{11}} & \dots & \frac{-a_{1n}}{a_{11}} \\ \frac{-a_{21}}{a_{22}} & 0 & \dots & \frac{-a_{2n}}{a_{22}} \\ \vdots & & \ddots & \vdots \\ \frac{-a_{n1}}{a_{nn}} & \frac{-a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix}, c = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix}$$

Given initial approximation $x^{(0)}$, we generate the sequence of vectors $\{x^{(k)}\}_{k=0}^{\infty}$ by computing:

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

$$for\ \ i = 1,2,\dots,n \text{ and } k \in \mathbb{N}^* \qquad (3.8)$$

Also, we can derive formula (3.7) by splitting a matrix A into its diagonal and off-diagonal parts.

Let D be the diagonal matrix where entries are those of matrix A, let $-L$ be strictly lower triangular matrix and $-U$ be the strictly upper triangular part of matrix A [12, 33].

With this notation, matrix A is spilt into:

$$A = D - L - U \qquad (3.9)$$

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

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

Substituting (3.9) into (3.1) yields

$$(D - L - U)\mathbf{x} = \boldsymbol{b} \qquad (3.10)$$

Equation (3.10) can be written as:

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

If D^{-1}exists, then:

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

This result is the matrix form of the Jacobi scheme:

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

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

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

So,

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

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 [29,10].

3.2 Gauss-Seidel Method

This iterative method is used for solving a square linear system $A\mathbf{x} = \boldsymbol{b}$ which is similar to the Jacobi method. For 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 $x_i{}^{(k+1)}$ reached. 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 [7].

To derive the general form of Gauss-Seidel method, consider the following $n \times n$ linear system:

$$x_1^{(k)} = \frac{-a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} - \cdots - \frac{a_{1n}}{a_{11}} x_n^{(k-1)} + \frac{b_1}{a_{11}}$$

$$x_2^{(k)} = \frac{-a_{21}}{a_{22}} x_1^{(k)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} - \cdots - \frac{a_{2n}}{a_{22}} x_n^{(k-1)} + \frac{b_2}{a_{22}}$$

$$\vdots \qquad (3.11)$$

$$x_n^{(k)} = \frac{-a_{n1}}{a_{nn}} x_1^{(k)} - \frac{a_{n2}}{a_{nn}} x_2^{(k)} - \cdots - \frac{a_{nn-1}}{a_{nn}} x_{n-1}^{(k)} + \frac{b_n}{a_{nn}}$$

Given initial approximation $x^{(0)}$, we generate the sequence of vectors $\{x^{(k)}\}_{k=0}^{\infty}$ by computing:

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

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

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

$$for\ \ i = 1,2,\ldots,n \text{ and } k \in \mathbb{N}^* \qquad (3.12)$$

Rearranging equation (3.10) yields

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

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

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

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

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

using $T_g = (D-L)^{-1}U$ and $c_g = (D-L)^{-1}b$, we obtain the Gauss-Seidel technique of the form [10,29]:

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

3.3 Successive Over Relaxation (SOR) Method

To use successive over relaxation method, the coefficient matrix must be symmetric and positive definite. For any real positive number $\omega \in (0,2)$ is called the relaxation parameter. If $\omega \in (0,1)$, then the method is called successive under relaxation. This method can be used to achieve convergence of systems that are not convergent by Gauss-Seidel method. On the other hand, if $\omega \in (1,2)$, then the method is called successive over relaxation method. Here, accelerated convergence of a linear systems that are already convergent by Gauss-Seidel method. If $\omega = 1$, we get Gauss-Seidel method [28 ,29].

Gauss-Seidel method in (3.11), will be used to derive the general formula of successive over relaxation method.

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

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

Define the difference

$$\Delta x_i = x_i^{(k)} - x_i^{(k-1)} \tag{3.13}$$

Rearranging equation (3.13), we get

$$x_i^{(k)} = x_i^{(k-1)} + \Delta x_i \tag{3.14}$$

Multiplying Δx_i in (3.14) by relaxation parameter ω yileds

$$x_i^{(k)} = x_i^{(k-1)} + \omega \Delta x_i \tag{3.15}$$

Rearranging equation (3.15) to get

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

$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \omega x_i^{(k)} \tag{3.16}$$

Now, put (3.12) into (3.16) yields:

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

$$for\ a_{ii} \neq 0, i = 1,2, \dots, n \text{ and } k \in \mathbb{N}^* \tag{3.17}$$

this formula is called (SOR)

Also, to write (3.17) in the matrix form

Since $a_{ii} \neq 0$ multiply each side of (3.17) by a_{ii}

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

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

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

$$(D+\omega L)\mathbf{x}^{(k)} = \big((1-\omega)D + \omega U\big)\mathbf{x}^{(k-1)} + \omega b_i \qquad (3.18)$$

if $(D+\omega L)^{-1}$ exist, multiply both sides of (3.18) by $(D+\omega L)^{-1}$ we get:

$$\mathbf{x}^{(k)} = (D+\omega L)^{-1}\big((1-\omega)D + \omega U\big)\mathbf{x}^{(k-1)} + \omega(D+\omega L)^{-1}b_i \quad (3.18)$$

in (3.18) let $T_\omega = (D+\omega L)^{-1}\big((1-\omega)D + \omega U\big)$ and $c_\omega = \omega(D+\omega L)^{-1}b_i$ we get:

$$\mathbf{x}^{(k)} = T_\omega\mathbf{x}^{(k-1)} + c_\omega \qquad (3.20)$$

3.4 Conjugate Gradient Method

The conjugate gradient method is an iterative method that is used to approximate the exact solution of the linear system $A\mathbf{x} = b$, where the coefficient matrix A must be symmetric and positive definite.

We denote the initial guess $\mathbf{x}_0$, and we may assume without loss of generality that $\mathbf{x} = \mathbf{x}_0$, otherwise be given. Starting with $\mathbf{x}_0$ we search for the solution and each iteration we need a metric to tell us whether we are

closer to the exact solution is absolute. The unique solution minimizes the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \boldsymbol{b}^T\mathbf{x} \quad \mathbf{x} \in \mathbb{R}^n$$

And for simplicity, we will take the conjugate gradient method as algorithm [5,29].

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 r_0 as follows:

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

Step 3: Let the initial direction vector $\boldsymbol{p_0} = \boldsymbol{r_0}$, that is, the negative of the gradient of the quadratic function:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \boldsymbol{b}^T\mathbf{x} \text{ at } \mathbf{x} = \mathbf{x_0}.$$

we see that $\boldsymbol{p}_k$ will change in each iteration.

Step 4: Compute the scalars α_k's using the formula:

$$\alpha_k = \frac{\boldsymbol{r_k^T}\ \boldsymbol{r_k}}{\boldsymbol{p_k^T}A\ \ \boldsymbol{p_k}}\ , \forall k = 0,1,2,\dots,n-1.$$

Step 5: Compute the first iteration $\mathbf{x_1}$ using the formula:

$$\mathbf{x_1} = \mathbf{x_0} + \alpha_0\boldsymbol{p_0}$$

Step 6: Compute the residual vectors r_k 's using the formula:

$$\boldsymbol{r_{k+1}} = \boldsymbol{r_k} - \alpha_k A\boldsymbol{p_k}, \forall k = 0,1,2,\dots,n-1.$$

Step 7: Compute the scalars β_k 's using the formula:

$$\beta_k = \frac{\boldsymbol{r_{k+1}^T}\ \boldsymbol{r_{k+1}}}{\boldsymbol{r_k^T}\ \boldsymbol{r_k}}, \forall k = 0,1,2,\dots,n-1.$$

Step 8: Compute the direction vectors $\boldsymbol{p_k}$'s using the formula:

$$\boldsymbol{p_{k+1}} = \boldsymbol{r_{k+1}} + \beta_k \boldsymbol{p_k}, \forall k = 0,1,2,\dots,n-1.$$

Step 9: Compute the iterations $\mathbf{x_k}$ using the formula:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \boldsymbol{p_k}, \forall k = 1,2,\dots,n-1.$$

3.5 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 that reached to the solution. In any computational problem, we'll 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 $\mathbf{x}$ and the approximate solution $\mathbf{x}^{(\boldsymbol{k})}$. But we cannot compute its value because we do not know the exact solution.

Instead, we will deal with the estimated error, which equals the difference between the approximate solution $\mathbf{x}^{(\boldsymbol{k})}$ and the next approximate solution $\mathbf{x}^{(\boldsymbol{k+1})}$ [7].

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

Suppose **x** is the exact solution of the following linear system:

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

This can be written in an equivalent form as:

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

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

The idea of the iterative methods is to generate a sequence of vectors $\{\mathbf{x}^{(\boldsymbol{k})}\}_{k=0}^{\infty}$ that converges to the exact solution **x** of the linear system $A\mathbf{x} = \mathbf{b}$. (Note: Each vector in the sequence is an approximation to the exact solution) [5].

Before going through convergence of the iterative methods, we need some definitions:

Definition 3.1 [7]

An $n \times n$ matrix A is positive definite if A is symmetric matrix and $C^T AC > 0$ for any non-zero n-dimensional column vector C.

Definition 3.2 [7]

Let $\lambda_1, \lambda_2, \dots, \lambda_n$, be eigenvalues of the matrix $A_{n\times n}$. Then the spectral radius $\rho(A)$ defined as: $\rho(A) = max_{1\leq i\leq n}\{\lambda_1, \lambda_2, \dots, \lambda_n\}$.

Definition 3.3 [7]

The l_2 and l_∞ norms for the vector $x = \{x_1, x_2, \dots, x_n\}^t$ are defined by $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$ and $\|x\|_\infty = \max_{1 \le i \le n}\{x_i\}$.

3.5.1 Convergence of Jacobi and Gauss-Seidel Iterative Methods

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

Theorem 3.1 [29]

For any initial approximation, a sequence of vectors $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ converges to the exact solution **x** if and only if the spectral radius of the square matrix T, $\rho(T) < 1$. (T is the matrix as in (3.2) form).

Theorem 3.2 [29]

If the coefficient matrix A for the linear system (3.1) is strictly diagonally dominant, then the sequence of vectors $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ generated by the Jacobi and Gauss-Seidel Iterative techniques converges to the unique solution of that system.

Theorem 3.3 [7]

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

3.5.2 Convergence of SOR iterative Method

Theorem 3.4 "Ostrowski-Reich" [7]

If the coefficient matrix A of the linear system (3.20) 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 $\mathrm{x}^{(0)}$.

3.5.3 Convergence of Conjugate Gradient Method

Theorem 3.5 [24]

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 $A\mathbf{x} = \boldsymbol{b}$ of n variables in at most n steps for any choice of initial approximation vector $\mathbf{x}^{(0)}$.

Proof:[24]

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

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

$\mathbf{x} - \mathbf{x}^{(0)} = a_0\boldsymbol{p_0} + a_1\boldsymbol{p_1} + a_2\boldsymbol{p_2} + \cdots + a_{n-1}\boldsymbol{p_{n-1}}$, where a_i 's $\in \mathbb{R}$.

Multiplying both sides of the last expression by $p_j^T A$, we obtain

$$\boldsymbol{p_j^T} A(\mathbf{x} - \mathbf{x}^{(0)}) = \boldsymbol{p_j^T} A(a_0\boldsymbol{p_0} + a_1\boldsymbol{p_1} + a_2\boldsymbol{p_2} + \cdots + a_{n-1}\boldsymbol{p_{n-1}})$$

Simplify the above expression, we get

$$\boldsymbol{p_j^T} A\mathbf{x} - \boldsymbol{p_j^T} A\mathbf{x}^{(0)} = a_0\boldsymbol{p_j^T} A\boldsymbol{p_0} + a_1\boldsymbol{p_j^T} A\boldsymbol{p_1} + a_2\boldsymbol{p_j^T} A\boldsymbol{p_2} + \cdots + a_{n-1}\boldsymbol{p_j^T} A\boldsymbol{p_{n-1}}$$

but $\boldsymbol{b} = A\mathbf{x}$, $\boldsymbol{r_0} = \boldsymbol{b} - A\mathbf{x_0}$, and $\boldsymbol{p_j^T} A\boldsymbol{p_i} = 0, \forall i \neq j$. So, it becomes:

$$\boldsymbol{p_j^T r_0} = \boldsymbol{a_j}\, \boldsymbol{p_j^T} A\boldsymbol{p_j}$$

Thus,

$$\boldsymbol{a_j} = \frac{\boldsymbol{p_j^T r_0}}{\boldsymbol{p_j^T A p_j}} \tag{3.21}$$

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

$$\boldsymbol{\alpha_j} = \frac{\boldsymbol{r_j^T \ r_j}}{\boldsymbol{p_j^T A \ p_j}}, \forall j = 0,1,2, \dots, n-1.$$

$$\mathbf{x_j} = \mathbf{x_0} + \boldsymbol{a_0 p_0} + \boldsymbol{a_1 p_1} + \boldsymbol{a_2 p_2} + \cdots + \boldsymbol{a_{j-1} p_{j-1}}$$

Multiply both sides of the last equation by $\boldsymbol{p_j^T A}$

$$\boldsymbol{p_j^T A x_j} = \boldsymbol{p_j^T A}(\mathbf{x_0} + a_0\boldsymbol{p_0} + a_1\boldsymbol{p_1} + a_2\boldsymbol{p_2} + \cdots + a_{j-1}\boldsymbol{p_{j-1}})$$

$$= \boldsymbol{p_j^T A x_0} + \boldsymbol{p_j^T A}(a_0\boldsymbol{p_0} + a_1\boldsymbol{p_1} + a_2\boldsymbol{p_2} + \cdots + a_{j-1}\boldsymbol{p_{j-1}})$$

$$= \boldsymbol{p_j^T A x_0} + 0$$

The above can be written as:

$$\boldsymbol{p_j^T A x_j} - \boldsymbol{p_j^T A x_0} = 0$$

or

$$\boldsymbol{p_j^T A}\left(\mathbf{x_j} - \mathbf{x_0}\right) = 0$$

Therefore,

$$\boldsymbol{p_j^T r_0} = \boldsymbol{p_j^T A}(\mathbf{x} - \mathbf{x^{(0)}}) = \boldsymbol{p_j^T A}(\mathbf{x} - \mathbf{x_j} + \mathbf{x_j} - \mathbf{x^{(0)}})$$

$$\boldsymbol{p_j^T r_0} = \boldsymbol{p_j^T A}\left(\mathbf{x} - \mathbf{x_j}\right) + \boldsymbol{p_j^T A}\left(\mathbf{x_j} - \mathbf{x^{(0)}}\right)$$

$$= \boldsymbol{p_j^T}\left(\boldsymbol{A x} - \boldsymbol{A x_j}\right) + 0$$

$$= \boldsymbol{p_j^T}(\boldsymbol{b} - \boldsymbol{A x_j}) = \boldsymbol{p_j^T r_j}$$

Now, put $\boldsymbol{p_j^T r_0} = \boldsymbol{p_j^T r_j}$ in equation (3.21), then we get:

$$\boldsymbol{a_j} = \frac{\boldsymbol{p_j^T r_j}}{\boldsymbol{p_j^T A p_j}} = \boldsymbol{\alpha_j}.$$

Chapter Four
Numerical Results

In this chapter, the finite difference and finite element methods are used to solve homogeneous and inhomogeneous one-dimensional heat equation subject to different types of boundary conditions: Dirichlet, Neumann and Robin's. Moreover, a comparison is carried out between the aforementioned iterative methods.

Example 4.1: Consider the one-dimensional heat equation

$$u_t = 1.25u_{xx}, \qquad 0 \le x \le 10, t \ge 0$$

Subject to the initial condition $u(x,0) = 0$ and boundary conditions $u(0,t) = 100, u(10,t) = 50.$

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

First, we start with a partition for the domain by divide x-axis into equal steps $h = \frac{b-a}{n} = \frac{10-0}{4} = 2.5,$ also we divide t-axis into equal steps $k = 2$ as shown in Figure (4.1).

Now, we define the mesh points (x_i, t_j) as follow:

$$x_i = a + ih \qquad , i = 0,1,2,3,4$$

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

For $i = 0$, $x_0 = 0 + 0 \times 2.5 = 0$

$i = 1$, $x_1 = 0 + 1 \times 2.5 = 2.5$

$i = 2$, $x_2 = 0 + 2 \times 2.5 = 5$

$i = 3$, $x_3 = 0 + 3 \times 2.5 = 7.5$

$i = 4$, $x_4 = 0 + 4 \times 2.5 = 10$

And for $j = 0$, $t_0 = 0 + 0 \times 2 = 0$

$j = 1$, $t_1 = 0 + 1 \times 2 = 2$

$j = 2$, $t_2 = 0 + 2 \times 2 = 4$

$j = 3$, $t_3 = 0 + 3 \times 2 = 6$

These partitions are illustrated in Figure 4.1

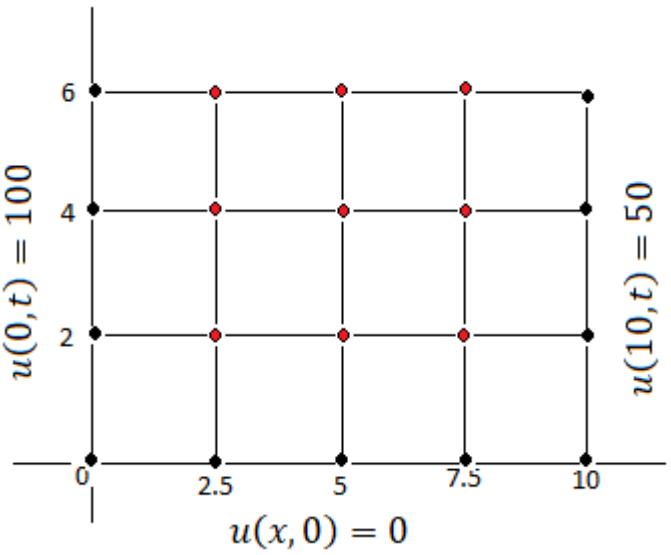

Figure (4.1): Discretization of the domain for example 4.1

The black points are known boundary points and the red points (interior) points are unknown which are to be approximate.

using the formula:

$$u_{i,j-1} = (1 + 2\ \lambda)u_{i,j} + \ \lambda\left(u_{i+1,j} + u_{i-1,j}\right)$$

to approximate the interior points.

For:

$$(i = 1, j = 1),\ u_{1,0} = (1 + 2\ \lambda)u_{1,1} + \ \lambda\left(u_{2,1} + u_{0,1}\right) \qquad (1)$$

$$(i = 2, j = 1),\ u_{2,0} = (1 + 2\ \lambda)u_{2,1} + \ \lambda\left(u_{3,1} + u_{1,1}\right) \qquad (2)$$

$$(i = 3, j = 1),\ u_{3,0} = (1 + 2\ \lambda)u_{3,1} + \ \lambda\left(u_{4,1} + u_{2,1}\right) \qquad (3)$$

$$(i = 1, j = 2),\ u_{1,1} = (1 + 2\ \lambda)u_{1,2} + \ \lambda\left(u_{2,2} + u_{0,2}\right) \qquad (4)$$

$$(i = 2, j = 2),\ u_{2,1} = (1 + 2\ \lambda)u_{2,2} + \ \lambda\left(u_{3,2} + u_{1,2}\right) \qquad (5)$$

$$(i = 3, j = 2),\ u_{3,1} = (1 + 2\ \lambda)u_{3,2} + \ \lambda\left(u_{4,2} + u_{2,2}\right) \qquad (6)$$

$$(i = 1, j = 3),\ u_{1,2} = (1 + 2\ \lambda)u_{1,3} + \ \lambda\left(u_{2,3} + u_{0,3}\right) \qquad (7)$$

$$(i = 2, j = 3),\ u_{2,2} = (1 + 2\ \lambda)u_{2,3} + \ \lambda\left(u_{3,3} + u_{1,3}\right) \qquad (8)$$

$$(i = 3, j = 3),\ u_{3,2} = (1 + 2\ \lambda)u_{3,3} + \ \lambda\left(u_{4,3} + u_{2,3}\right) \qquad (9)$$

by put the initial and boundary conditions in equations 1to 9 and use the notation ($u_{1,1} = u_1, u_{2,1} = u_2, u_{3,1} = u_3, u_{1,2} = u_4, u_{2,2} = u_5, u_{3,2} = u_6\ , u_{1,3} = u_7,\ \ u_{2,3} = u_8, u_{3,3} = u_9$) yields:

$$1.8u_1 - 0.4u_2 40 \quad (1)$$

$$1.8u_2 - 0.4u_3 - 0.4u_1 = 0 \quad (2)$$

$$1.8u_3 - 0.4u_2 = 20 \quad (3)$$

$$1.8u_4 - 0.4u_5 - u_1 = 40 \quad (4)$$

$$1.8u_5 - 0.4u_6 - 0.4u_4 - u_2 = 0 \quad (5)$$

$$1.8u_6 - 0.4u_5 - u_3 = 20 \quad (6)$$

$$1.8u_7 - 0.4u_8 - u_4 = 40 \quad (7)$$

$$1.8u_8 - 0.4u_9 - 0.4u_7 - u_5 = 0 \quad (8)$$

$$1.8u_9 - 0.4u_8 - u_6 = 20 \quad (9)$$

The above equations can be expressed in matrix form as:

$$\begin{bmatrix} 1.8 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.4 & 1.8 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.4 & 1.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1.8 & -0.4 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0.4 & 1.8 & -0.4 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -0.4 & 1.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1.8 & -0.4 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -0.4 & 1.8 & -0.4 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -0.4 & 1.8 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix} \quad (4.1)$$

Using $\mathbf{u} = \boldsymbol{A^{-1}b}$ we obtain the exact solution:

$\mathbf{u} = (\ 24.049, 8.2192, 12.937, 39.661, 18.352, 22.377, 50.485, 28.031, 29.772)^T$

We can also solve the linear system (4.1) by the following iterative techniques:

Jacobi Method

The Jacobi method given by the sequence

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

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

where n is the number of the unknown variables

$$u_1^{(k)} = \frac{0.4}{1.8}u_2^{(k-1)} + \frac{40}{1.8}$$

$$u_2^{(k)} = \frac{0.4}{1.8}u_1^{(k-1)} + \frac{0.4}{1.8}u_3^{(k-1)}$$

$$u_3^{(k)} = \frac{0.4}{1.8}u_2^{(k-1)} + \frac{20}{1.8}$$

$$u_4^{(k)} = \frac{0.4}{1.8}u_5^{(k-1)} + \frac{1}{1.8}u_1^{(k-1)} + \frac{40}{1.8}$$

$$u_5^{(k)} = \frac{0.4}{1.8}u_6^{(k-1)} + \frac{0.4}{1.8}u_5^{(k-1)} + \frac{1}{1.8}u_2^{(k-1)} \qquad (4.2)$$

$$u_6^{(k)} = \frac{0.4}{1.8}u_5^{(k-1)} + \frac{1}{1.8}u_3^{(k-1)} + \frac{20}{1.8}$$

$$u_7^{(k)} = \frac{0.4}{1.8}u_8^{(k-1)} + \frac{1}{1.8}u_3^{(k-1)} + \frac{40}{1.8}$$

$$u_8^{(k)} = \frac{0.4}{1.8}u_9^{(k-1)} + \frac{0.4}{1.8}u_7^{(k-1)} + \frac{1}{1.8}u_5^{(k-1)}$$

$$u_9^{(k)} = \frac{0.4}{1.8}u_8^{(k-1)} + \frac{1}{1.8}u_6^{(k-1)} + \frac{20}{1.8}$$

Consider the initial solution is $u^{(0)} = (0,0,0,0,0,0,0,0,0)^T$, so we use the initial solution in system (4.2) to find the first iteration $u^{(1)}$

$$u_1^{(1)} = \frac{0.4}{1.8}u_2^{(0)} + \frac{40}{1.8} = 22.22$$

$$u_2^{(1)} = \frac{0.4}{1.8}u_1^{(0)} + \frac{0.4}{1.8}u_3^{(0)} = 0$$

$$u_3^{(1)} = \frac{0.4}{1.8}u_2^{(0)} + \frac{20}{1.8} = 11.11$$

$$u_4^{(1)} = \frac{0.4}{1.8}u_5^{(0)} + \frac{1}{1.8}u_1^{(0)} + \frac{40}{1.8} = 22.22$$

$$u_5^{(1)} = \frac{0.4}{1.8}u_6^{(0)} + \frac{0.4}{1.8}u_5^{(0)} + \frac{1}{1.8}u_2^{(0)} = 0$$

$$u_6^{(1)} = \frac{0.4}{1.8}u_5^{(0)} + \frac{1}{1.8}u_3^{(0)} + \frac{20}{1.8} = 11.11$$

$$u_7^{(1)} = \frac{0.4}{1.8}u_8^{(0)} + \frac{1}{1.8}u_3^{(0)} + \frac{40}{1.8} = 22.22$$

$$u_8^{(1)} = \frac{0.4}{1.8}u_9^{(0)} + \frac{0.4}{1.8}u_7^{(0)} + \frac{1}{1.8}u_5^{(0)} = 0$$

$$u_9^{(1)} = \frac{0.4}{1.8}u_8^{(0)} + \frac{1}{1.8}u_6^{(0)} + \frac{20}{1.8} = 11.11$$

The first iteration gives

$$u^{(1)} = (22.22,0,11.11,22.22,0,11.11,22.22,0,11.11)^T$$

Likewise, after 24 iterations we obtain the approximate solution:

$$\boldsymbol{u} = (24.049,8.219,12.938,39.661,18.352,22.377,50.485,28.030,29.772)^T$$

Number of iterations	The error
24	5.11493×10^{-008}

The Matlab code for the Jacobi iterative method can be formed in Appendix A.

Gauss-Seidel Method

It is given by the sequence (3.11)

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

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

where n is the number of the unknown variable.

$$u_1^{(k)} = \frac{0.4}{1.8} u_2^{(k-1)} + \frac{40}{1.8}$$

$$u_2^{(k)} = \frac{0.4}{1.8} u_1^{(k)} + \frac{0.4}{1.8} u_3^{(k-1)}$$

$$u_3^{(k)} = \frac{0.4}{1.8} u_2^{(k)} + \frac{20}{1.8} \qquad (4.3)$$

$$u_4^{(k)} = \frac{0.4}{1.8} u_5^{(k-1)} + \frac{1}{1.8} u_1^{(k)} + \frac{40}{1.8}$$

$$u_5^{(k)} = \frac{0.4}{1.8} u_6^{(k-1)} + \frac{0.4}{1.8} u_4^{(k)} + \frac{1}{1.8} u_2^{(k)}$$

$$u_6^{(k)} = \frac{0.4}{1.8} u_5^{(k)} + \frac{1}{1.8} u_3^{(k)} + \frac{20}{1.8}$$

$$u_7^{(k)} = \frac{0.4}{1.8} u_8^{(k-1)} + \frac{1}{1.8} u_3^{(k)} + \frac{40}{1.8}$$

$$u_8^{(k)} = \frac{0.4}{1.8} u_9^{(k-1)} + \frac{0.4}{1.8} u_7^{(k)} + \frac{1}{1.8} u_5^{(k)}$$

$$u_9^{(k)} = \frac{0.4}{1.8} u_8^{(k)} + \frac{1}{1.8} u_6^{(k)} + \frac{20}{1.8}$$

Choose the initial solution as $u^{(0)} = (0,0,0,0,0,0,0,0,0)^T$, then we find the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = \frac{0.4}{1.8} u_2^{(0)} + \frac{40}{1.8} = 22.22$$

$$u_2^{(1)} = \frac{0.4}{1.8} u_1^{(1)} + \frac{0.4}{1.8} u_3^{(0)} = \frac{0.4}{1.8} \times 22.22 = 4.94$$

$$u_3^{(1)} = \frac{0.4}{1.8} u_2^{(1)} + \frac{20}{1.8} = \frac{0.4}{1.8} \times 494 + \frac{20}{1.8} = 12.208$$

$$u_4^{(1)} = \frac{0.4}{1.8}u_5^{(0)} + \frac{1}{1.8}u_1^{(1)} + \frac{40}{1.8} = \frac{22.22}{1.8} + \frac{40}{1.8} = 34.566$$

$$u_5^{(1)} = \frac{0.4}{1.8}u_6^{(0)} + \frac{0.4}{1.8}u_4^{(1)} + \frac{1}{1.8}u_2^{(1)} = \frac{4.94}{1.8} + \frac{0.4 \times 34.566}{1.8} = 10.426$$

$$u_6^{(1)} = \frac{0.4}{1.8}u_5^{(1)} + \frac{1}{1.8}u_3^{(1)} + \frac{20}{1.8} = \frac{12.208}{1.8} + \frac{0.4 \times 10.426}{1.8} + \frac{20}{1.8}$$

$$= 21.210$$

$$u_7^{(1)} = \frac{0.4}{1.8}u_8^{(0)} + \frac{1}{1.8}u_3^{(1)} + \frac{40}{1.8} = \frac{34.566}{1.8} + \frac{40}{1.8} = 41.425$$

$$u_8^{(1)} = \frac{0.4}{1.8}u_9^{(0)} + \frac{0.4}{1.8}u_7^{(1)} + \frac{1}{1.8}u_5^{(1)} = \frac{10.426}{1.8} + \frac{0.4 \times 41.425}{1.8} = 14.997$$

$$u_9^{(1)} = \frac{0.4}{1.8}u_8^{(1)} + \frac{1}{1.8}u_6^{(1)} + \frac{20}{1.8} = \frac{20.21}{1.8} + \frac{0.4 \times 14.997}{1.8} + \frac{20}{1.8} = 25.671$$

The first iteration gives

$$u^{(1)} = (22.22,4.94,12.208,34.566,10.426,20.12,41.425,14.997,25.671)^T$$

Likewise, after 24 iterations we obtain the approximate solution:

$$u = (24.049,8.219,12.938,39.661,18.352,22.377,50.485,28.030,29.772)^T$$

Number of iterations	The error
24	$5.114938 \times 10^{-008}$

The Matlab code for the Gauss-Seidel iterative method can be formed in Appendix B.

Successive Over Relaxation (SOR) Method

The SOR method is given by the sequence (3.16)

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

$$for\ a_{ii} \neq 0, i = 1,2, \dots, n \text{ and } k \in \mathbb{N}^*$$

Here we will choose the relaxation factor $\omega = 1.3$.

The Gauss-Seidel equations are:

$$u_1^{(k)} = \frac{0.4}{1.8}u_2^{(k-1)} + \frac{40}{1.8}$$

$$u_2^{(k)} = \frac{0.4}{1.8}u_1^{(k)} + \frac{0.4}{1.8}u_3^{(k-1)}$$

$$u_3^{(k)} = \frac{0.4}{1.8}u_2^{(k)} + \frac{20}{1.8}$$

$$u_4^{(k)} = \frac{0.4}{1.8}u_5^{(k-1)} + \frac{1}{1.8}u_1^{(k)} + \frac{40}{1.8}$$

$$u_5^{(k)} = \frac{0.4}{1.8}u_6^{(k-1)} + \frac{0.4}{1.8}u_4^{(k)} + \frac{1}{1.8}u_2^{(k)}$$

$$u_6^{(k)} = \frac{0.4}{1.8}u_5^{(k)} + \frac{1}{1.8}u_3^{(k)} + \frac{20}{1.8}$$

$$u_7^{(k)} = \frac{0.4}{1.8}u_8^{(k-1)} + \frac{1}{1.8}u_3^{(k)} + \frac{40}{1.8}$$

$$u_8^{(k)} = \frac{0.4}{1.8}u_9^{(k-1)} + \frac{0.4}{1.8}u_7^{(k)} + \frac{1}{1.8}u_5^{(k)}$$

$$u_9^{(k)} = \frac{0.4}{1.8}u_8^{(k)} + \frac{1}{1.8}u_6^{(k)} + \frac{20}{1.8}$$

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

$$u_1^{(k)} = (1 - 1.3)u_1^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_2^{(k-1)} + \frac{40}{1.8}\right]$$

$$u_2^{(k)} = (1 - 1.3)u_2^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_1^{(k)} + \frac{0.4}{1.8}u_3^{(k-1)}\right]$$

$$u_3^{(k)} = (1 - 1.3)u_3^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_2^{(k)} + \frac{20}{1.8}\right]$$

$$u_4^{(k)} = (1-1.3)u_4^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_5^{(k-1)} + \frac{1}{1.8}u_1^{(k)} + \frac{40}{1.8}\right]$$

$$u_5^{(k)} = (1-1.3)u_5^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_6^{(k-1)} + \frac{0.4}{1.8}u_4^{(k)} + \frac{1}{1.8}u_2^{(k)}\right]$$

$$u_6^{(k)} = (1-1.3)u_6^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_5^{(k)} + \frac{1}{1.8}u_3^{(k)} + \frac{20}{1.8}\right] \qquad (4.4)$$

$$u_7^{(k)} = (1-1.3)u_7^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_8^{(k-1)} + \frac{1}{1.8}u_3^{(k)} + \frac{40}{1.8}\right]$$

$$u_8^{(k)} = (1-1.3)u_8^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_9^{(k-1)} + \frac{0.4}{1.8}u_7^{(k)} + \frac{1}{1.8}u_5^{(k)}\right]$$

$$u_9^{(k)} = (1-1.3)u_9^{(k-1)} + 1.3\left[\frac{0.4}{1.8}u_8^{(k)} + \frac{1}{1.8}u_6^{(k)} + \frac{20}{1.8}\right]$$

Select the initial solution as $u^{(0)} = (0,0,0,0,0,0,0,0,0)^T$, then we obtain

$$u_1^{(1)} = (1-1.3)u_1^{(0)} + 1.3\left[\frac{0.4}{1.8}u_2^{(0)} + \frac{40}{1.8}\right] = 1.3 \times \left[\frac{40}{1.8}\right] = 28.888$$

$$u_2^{(1)} = (1-1.3)u_2^{(0)} + 1.3\left[\frac{0.4}{1.8}u_1^{(1)} + \frac{0.4}{1.8}u_3^{(0)}\right] = 1.3\left[\frac{0.4}{1.8} \times 28.888\right]$$

$$= 8.3454$$

$$u_3^{(1)} = (1-1.3)u_3^{(0)} + 1.3\left[\frac{0.4}{1.8}u_2^{(1)} + \frac{20}{1.8}\right] = 1.3\left[\frac{0.4 \times 8.3454}{1.8} + \frac{20}{1.8}\right]$$

$$= 16.8553$$

$$u_4^{(1)} = (1-1.3)u_4^{(0)} + 1.3\left[\frac{0.4}{1.8}u_5^{(0)} + \frac{1}{1.8}u_1^{(1)} + \frac{40}{1.8}\right]$$

$$= 1.3\left[\frac{1 \times 28.888}{1.8} + \frac{40}{1.8}\right] = 49.752$$

$$u_5^{(1)} = (1-1.3)u_5^{(0)} + 1.3\left[\frac{0.4}{1.8}u_6^{(0)} + \frac{0.4}{1.8}u_4^{(1)} + \frac{1}{1.8}u_2^{(1)}\right]$$

$$= 1.3\left[\frac{0.4 \times 49.752}{1.8} + \frac{8.3454}{1.8}\right] = 20.40$$

$$u_6^{(1)} = (1-1.3)u_6^{(0)} + 1.3\left[\frac{0.4}{1.8}u_5^{(1)} + \frac{1}{1.8}u_3^{(1)} + \frac{20}{1.8}\right]$$
$$= 1.3\left[\frac{0.4 \times 20.40}{1.8} + \frac{116.8553}{1.8} + \frac{20}{1.8}\right] = 32.5112$$

$$u_7^{(1)} = (1-1.3)u_7^{(0)} + 1.3\left[\frac{0.4}{1.8}u_8^{(0)} + \frac{1}{1.8}u_3^{(1)} + \frac{40}{1.8}\right]$$
$$= 1.3\left[\frac{16.8553}{1.8} + \frac{40}{1.8}\right] = 64.8216$$

$$u_8^{(1)} = (1-1.3)u_8^{(0)} + 1.3\left[\frac{0.4}{1.8}u_9^{(0)} + \frac{0.4}{1.8}u_7^{(1)} + \frac{1}{1.8}u_5^{(1)}\right]$$
$$= 1.3\left[\frac{0.4 \times 64.8216}{1.8} + \frac{20.40}{1.8}\right] = 33.4599$$

$$u_9^{(1)} = (1-1.3)u_9^{(0)} + 1.3\left[\frac{0.4}{1.8}u_8^{(1)} + \frac{1}{1.8}u_6^{(1)} + \frac{20}{1.8}\right]$$
$$= 1.3\left[\frac{0.4 \times 33.4599}{1.8} + \frac{32.5112}{1.8} + \frac{20}{1.8}\right] = 47.591$$

The first iteration $u^{(1)}$ is:

$$u^{(1)}$$
$$= (28.888, 8.3456, 16.8554, 49.7530, 20.4005, 32.5112, 64.8216, 33.4599, 47.5910)^T$$

Likewise, after 22 iterations we obtain the approximate solution:

$$u = (24.049, 8.219, 12.938, 39.661, 18.352, 22.377, 50.485, 28.030, 29.772)^T$$

Number of iterations	The error
22	8.64340×10^{-008}

The Matlab code for the SOR iterative method can be formed in Appendix C.

Conjugate Gradient Method

This algorithm can be implemented as follows:

Step1 Start with initial guess $u_0 = (0,0,0,0,0,0,0,0,0)^T$

Step2 Calculate the residual vector r_0 as follows:

$$r_0 = b - Au_0$$

$$r_0 = \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix} - \begin{bmatrix} 1.8 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.4 & 1.8 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.4 & 1.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1.8 & -0.4 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0.4 & 1.8 & -0.4 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -0.4 & 1.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1.8 & -0.4 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -0.4 & 1.8 & -0.4 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -0.4 & 1.8 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$r_0 = \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix}$$

Step 3: Let the initial direction vector $p_0 = r_0$. So

$$p_0 = \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix}$$

Step 4 compute the scalar $\alpha_k's$ by formula

$$\alpha_k = \frac{r_k^t r_k}{p_k^t A p_k}$$

for $k = 0$,

$$\alpha_0 = \frac{r_0^t r_0}{p_0^t A p_0}$$

$$\mathrm{r}_0^{\mathrm{t}}\mathrm{r}_0 = \begin{bmatrix} 40 & 0 & 20 & 40 & 0 & 20 & 40 & 0 & 20 \end{bmatrix} \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix} = 6000$$

$$\mathrm{p}_0^{\mathrm{t}}\mathrm{A}\mathrm{p}_0 = \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix}^{\boldsymbol{T}} \begin{bmatrix} 1.8 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.4 & 1.8 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.4 & 1.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1.8 & -0.4 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0.4 & 1.8 & -0.4 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -0.4 & 1.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1.8 & -0.4 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -0.4 & 1.8 & -0.4 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -0.4 & 1.8 \end{bmatrix} \begin{bmatrix} 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \\ 40 \\ 0 \\ 20 \end{bmatrix}$$

$$= 6800$$

Thus

$$\alpha_0 = \frac{6000}{6800} = 0.8823$$

Step5 Compute the first iteration u_1 by the formula

$$u_1 = u_0 + \alpha_0 p_0$$

$$u_1 = (0,0,0,0,0,0,0,0,0) + 0.8823(40,0,20,40,0,20,40,0,20)$$

the first approximation $u_1 = \begin{bmatrix} 35.2941 \\ 0 \\ 17.6471 \\ 35.2941 \\ 0 \\ 17.6471 \\ 35.2941 \\ 0 \\ 17.6471 \end{bmatrix}$

Likewise, after 10 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 24.0487 \\ 8.21917 \\ 12.9375 \\ 39.6609 \\ 18.3524 \\ 22.3769 \\ 50.4851 \\ 28.0306 \\ 29.7717 \end{bmatrix}$$

Number of iterations	The error
10	$2.224057 \times 10^{-009}$

Table 4.1: Comparison between the iterative methods for example 4.1

Methods / u	Jacobi Method	Gauss-Seidel Method	SOR Method	Conjugate Gradient
u_1	24.0487	24.0487	24.0487	24.0487
u_2	8.2192	8.21917	8.21917	8.21917
u_3	12.9376	12.9375	12.9375	12.9375
u_4	39.6609	39.6609	39.6609	39.6609
u_5	18.3524	18.3524	18.3524	18.3524
u_6	22.3770	22.3769	22.3769	22.3769
u_7	50.4851	50.4851	50.4851	50.4851
u_8	28.0307	28.0306	28.0306	28.0306
u_9	29.7718	29.7717	29.7717	29.7717
Number of iterations	24	24	22	10
Error	5.11493×10^{-008}	$5.114938 \times 10^{-008}$	8.64340×10^{-008}	$2.224057 \times 10^{-009}$

The Matlab code for the conjugate gradient iterative method can be formed in Appendix D.

Example 4.2: Consider the one-dimensional heat equation

$$u_t = 1.25u_{xx}, \qquad 0 \le x \le 10, t \ge 0$$

with initial condition $u(x,0) = 0$ and boundary conditions are

$u(0,t) = 100$, $u(10,t) = 50$ and the upper boundary condition $u(x,8) = 75$ to get rectangular domain

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

We will start with discretize the domain by finite element as shown in Figure (4.2).

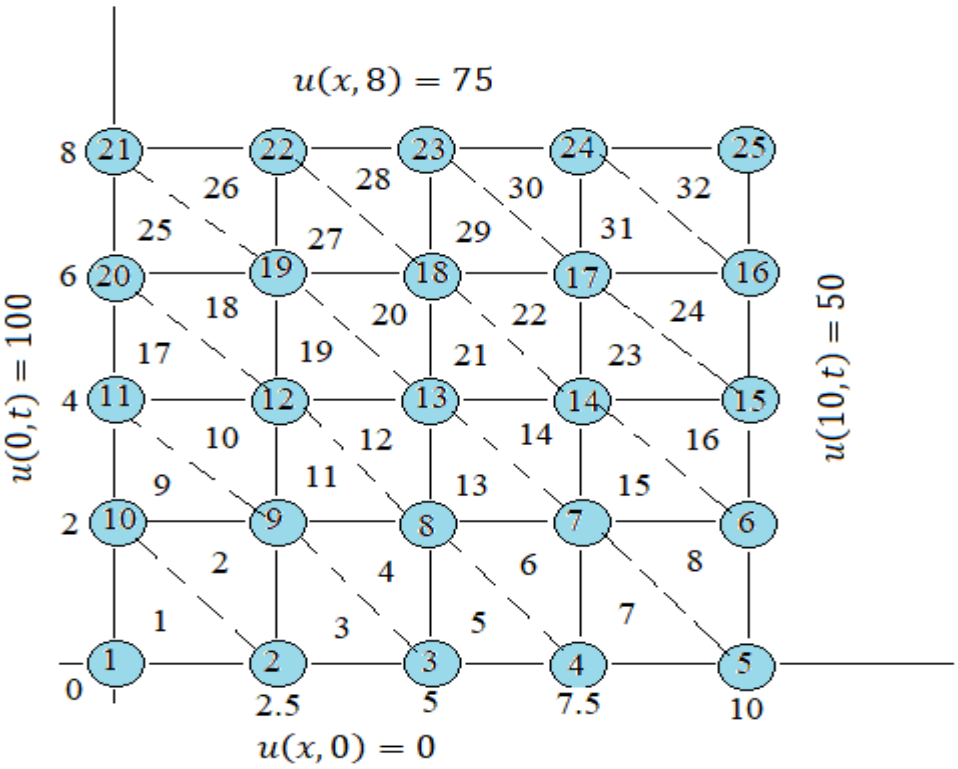

Figure (4.2): Discretization of the Domain by Finite Element Method

The region is divided into 32 equal triangular elements which are identified by encircled numbers 1 through 32 as indicated in Figure (4.4). In this discretization there are 25 global nodes. Now, we will write the coordinates for each node:

Node 1: (0 , 0), node 2: (2.5, 0), node 3: (5 , 0), node 4: (7.5 , 0)

node 5: (10 , 0), node 6: (10 , 2), node 7: (7.5 , 2), node 8: (5, 2)

node 9: (2.5 ,2), node 10: (0 , 2), node 11: (0 , 4), node 12:(2.5 , 4)

node 13: (5, 4), node 14: (7.5, 4), node 15: (10, 4), node 16: (10, 6)

node 17: (7.5, 6), node 18: (5, 6), node 19: (2.5, 6), node 20: (0, 6)

node 21: (0, 8), node 22: (2.5, 8), node 23: (5, 8), node 24: (7.5, 8) and node 25: (10, 8).

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

Table 4.4 shows that for each element we write its global nodes and their local node numbers and coordinates.

Table 4.2: 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**	$(x_2, t_2) = (2.5, 0)$
	10	**3**	$(x_3, t_3) = (0, 2)$
Element 2	**2**	**1**	$(x_1, t_1) = (2.5, 0)$
	9	**2**	$(x_2, t_2) = (2.5, 2)$
	10	**3**	$(x_3, t_3) = (0, 2)$
	…	**….**	**…**
Element 31	**17**	**1**	$(x_1, t_1) = (7.5, 6)$
	16	**2**	$(x_2, t_2) = (10, 6)$
	24	**3**	$(x_3, t_3) = (7.5, 8)$
Element 32	**16**	**1**	$(x_1, t_1) = (10, 6)$
	25	**2**	$(x_2, t_2) = (10, 8)$
	24	**3**	$(x_3, t_3) = (7.5, 8)$

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

For element 1:

$P_1 = t_2 - t_3 = -2$ $\qquad$ $Q_1 = x_3 - x_2 = -2.5$

$P_2 = t_3 - t_1 = 2$ $\quad\quad$ $Q_2 = x_1 - x_3 = 0$

$P_3 = t_1 - t_2 = 0$ $\quad\quad$ $Q_3 = x_2 - x_1 = 2.5$

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

We use equation (2.32) to write the entries of the 3 × 3 element coefficient matrix, let us take element 1 as an example:

$C_{ij}^{(1)} = \frac{1}{4A}[P_iP_j + Q_iQ_j]$, for i, j =1,2,3, where:

$$A = \frac{1}{2}[P_2Q_3 - P_3Q_2] = \frac{1}{2}[2 \times 2.5 - 0 \times 0] = \frac{5}{2}$$

$$C_{11}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_1P_1 + Q_1Q_1] = \frac{1}{10}[2 \times 2 \pm 2.5 \times -2.5] = 1.025$$

$$C_{12}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_1P_2 + Q_1Q_2] = \frac{1}{10}[-2 \times 2 + 0 \times -2.5] = -4$$

$$C_{13}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_1P_3 + Q_1Q_3] = \frac{1}{10}[-2 \times 0 + 2.5 \times -2.5] = -0.625$$

$$C_{21}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_2P_1 + Q_2Q_1] = \frac{1}{10}[2 \times -2 + 0 \times -2.5] = -0.4$$

$$C_{22}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_2P_2 + Q_2Q_2] = \frac{1}{10}[2 \times 2 + 0 \times 0] = 0.4$$

$$C_{23}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_2P_3 + Q_2Q_3] = \frac{1}{10}[2 \times 0 + 0 \times 2.5] = 0$$

$$C_{31}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_3P_1 + Q_3Q_1] = \frac{1}{10}[0 \times -2 + 2.5 \times -2.5] = -0.625$$

$$C_{32}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_3P_2 + Q_3Q_2] = \frac{1}{10}[0 \times 2 + 2.5 \times 0] = 0$$

$$C_{33}{}^{(1)} = \frac{1}{4.\frac{5}{2}}[P_3P_3 + Q_3Q_3] = \frac{1}{10}[0 \times 0 + 2.5 \times 2.5] = 0.625$$

Thus, the 3 × 3 element coefficient matrix for element 1 is:

$$C^{(1)} = \begin{bmatrix} C_{11}{}^{(1)} & C_{12}{}^{(1)} & C_{13}{}^{(1)} \\ C_{21}{}^{(1)} & C_{22}{}^{(1)} & C_{23}{}^{(1)} \\ C_{31}{}^{(1)} & C_{32}{}^{(1)} & C_{33}{}^{(1)} \end{bmatrix} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

In a similar manner, we find the 3 × 3 element coefficient matrix for elements 2, 3, 4, …, 32.

$$C^{(2)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(3)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(4)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(5)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(6)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(7)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(8)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(9)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(10)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(11)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(12)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(13)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(14)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(15)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(16)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}, C^{(17)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}$$

$$C^{(31)} = \begin{bmatrix} 1.025 & -0.4 & -0.625 \\ -0.4 & 0.4 & 0 \\ -0.625 & 0 & 0.625 \end{bmatrix}, C^{(32)} = \begin{bmatrix} 0.625 & -0.625 & 0 \\ 0.625 & 1.025 & -0.4 \\ 0 & -0.4 & 0.4 \end{bmatrix}$$

The global coefficient matrix C assembled from the element coefficient matrices. Since there are 25 nodes, the global coefficient matrix will be a 25×25 matrix.

The one diagonal entries can be computed as follows:

For example:

$$C_{1,1} = C_{11}{}^{(1)} = 1.025$$

$$C_{2,2} = C_{22}{}^{(1)} + C_{11}{}^{(2)} + C_{11}{}^{(3)} = 2.05$$

$$C_{3,3} = C_{22}{}^{(3)} + C_{11}{}^{(4)} + C_{11}{}^{(5)} = 2.05$$

$$C_{7,7} = C_{22}{}^{(6)} + C_{33}{}^{(7)} + C_{33}{}^{(8)} + C_{22}{}^{(13)} + C_{11}{}^{(15)} + C_{11}{}^{(14)} = 4.1$$

$$\vdots$$

$$C_{25,25} = C_{22}{}^{(32)} = 1.025$$

For the off-diagonal entries, for example $C_{9,12}$, the global link 9−12 corresponds to local link 1−3 of element 11 and local link 1−2 of element 10 as shown in Figure (4.4) and hence

$$C_{9,12} = C_{13}{}^{(11)} + C_{12}{}^{(10)} = -1.25$$

We can compute the value of other off-diagonal entries in the same manner. We continue the process to obtain the global matrix.

$$\begin{bmatrix}
1.025 & -0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.4 & 2.05 & -0.4 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -0.4 & 2.05 & -0.4 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -0.4 & 2.05 & -0.4 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -0.4 & 1.025 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.625 & 2.05 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1.25 & 0 & -0.8 & 4.1 & -0.8 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1.25 & 0 & 0 & 0 & -0.8 & 4.1 & -0.8 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1.25 & 0 & 0 & 0 & 0 & 0 & -0.8 & 4.1 & -0.8 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 2.05 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 2.05 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & -0.8 & 4.1 & -0.8 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & -0.8 & 4.1 & -0.8 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & -0.8 & 4.1 & -0.8 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 2.05 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 2.05 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & -0.8 & 4.1 & -0.8 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & -0.8 & 4.1 & -0.8 & 0 & 0 & 0 & -1.25 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & -0.8 & 4.1 & -0.8 & 0 & -1.25 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 2.05 & -0.625 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 1.025 & -0.4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & -0.4 & 2.05 & -0.4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & -0.4 & 2.05 & -0.4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & -0.4 & 2.05 & -0.4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.4 & 1.025
\end{bmatrix}$$

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 4.3.

Table 4.3: Represents vector of functions at the boundary

Global node	Boundary conditions	The value of global node
1	$u = 100$ on left and $u = 0$ at bottom boundaries	The average of its boundary values $\frac{100 + 0}{2} = 50$
2	$u = 0$ on bottom boundary	0
3	$u = 0$ on bottom boundary	0
4	$u = 0$ on bottom boundary	0
5	$u = 0$ on bottom boundary and $u = 50$ at right boundary	The average of its boundary values $\frac{50 + 0}{2} = 25$
6	$u = 50$ on right boundary	50
10	$u = 100$ on left boundary	100
11	$u = 100$ on left boundary	100
15	$u = 50$ on right boundary	50
16	$u = 50$ on right boundary	50
20	$u = 100$ on left boundary	100
21	$u = 100$ on left boundary and $u = 75$ on upper boundary	The average of its boundary values $\frac{100 + 75}{2} = 87.5$
22	$u = 75$ on upper boundary	75
23	$u = 75$ on upper boundary	75
24	$u = 75$ on upper boundary	75
25	$u = 75$ on upper boundary and $u = 50$ at right boundary	The average of its boundary values $\frac{50 + 75}{2} = 62.5$

The vector u_n is:

$$u_n = (50,0,0,0,25,50,100,100,50,50,100,87.5,75,75,75,62.5)^T$$

Now, we will define 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.

$$c_{vv} = \begin{bmatrix} 4.1 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.8 & 4.1 & -0.8 & 0 & -1.25 & 0 & 0 & 0 & 0 \\ 0 & -0.8 & 4.1 & -1.25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.25 & 4.1 & -0.8 & 0 & 0 & 0 & -1.25 \\ 0 & -1.25 & 0 & -0.8 & 4.1 & -0.8 & 0 & -1.25 & 0 \\ -1.25 & 0 & 0 & 0 & -0.8 & 4.1 & -1.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.25 & 4.1 & -0.8 & 0 \\ 0 & 0 & 0 & 0 & -1.25 & 0 & -0.8 & 4.1 & -0.8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 4.1 \end{bmatrix}$$

$$c_{vn} = \begin{bmatrix} 0 & 0 & 0 & -1.25 & 0 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.25 & 0 & 0 & 0 & 0 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 0 & 0 & 0 & -1.25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 0 & -1.25 & 0 & 0 \end{bmatrix}$$

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

$${C_{vv}}^{-1} = \begin{bmatrix} 0.2941 & 0.0799 & 0.0223 & 0.0222 & 0.0595 & 0.1135 & 0.0401 & 0.0284 & 0.0123 \\ 0.0799 & 0.3164 & 0.0799 & 0.0595 & 0.1357 & 0.0595 & 0.0284 & 0.0524 & 0.0284 \\ 0.0223 & 0.0799 & 0.2941 & 0.1135 & 0.0595 & 0.0222 & 0.0123 & 0.0284 & 0.0401 \\ 0.0222 & 0.0595 & 0.1135 & 0.3342 & 0.1082 & 0.0346 & 0.0222 & 0.0595 & 0.1135 \\ 0.0595 & 0.1357 & 0.0595 & 0.1082 & 0.3689 & 0.1082 & 0.0595 & 0.1357 & 0.0595 \\ 0.1135 & 0.0595 & 0.0222 & 0.0346 & 0.1082 & 0.3342 & 0.1135 & 0.1395 & 0.0222 \\ 0.0401 & 0.0284 & 0.0123 & 0.0222 & 0.0595 & 0.1135 & 0.2941 & 0.0595 & 0.0223 \\ 0.0284 & 0.0524 & 0.0284 & 0.0595 & 0.1357 & 0.0595 & 0.0799 & 0.3164 & 0.0799 \\ 0.0123 & 0.0284 & 0.0401 & 0.1135 & 0.0595 & 0.0222 & 0.0223 & 0.0799. & 0.2941 \end{bmatrix}$$

The vector u_v of unknown nodes can be found by using the formula:

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

$$u_v = \begin{bmatrix} node\ 7 \\ node\ 8 \\ node\ 9 \\ node\ 12 \\ node\ 13 \\ node\ 14 \\ node\ 17 \\ node\ 18 \\ node\ 19 \end{bmatrix} = \begin{bmatrix} 30.0272 \\ 30.3578 \\ 45.6648 \\ 66.3515 \\ 51.1306 \\ 47.0603 \\ 59.6071 \\ 64.7670 \\ 75.2446 \end{bmatrix}$$

Example 4.3: Consider the one-dimensional heat equation

$$u_t = u_{xx}\ , \qquad 0 \le x \le 1, t \ge 0$$

Subject to initial condition $u(x,0) = sin\pi x$ and boundary conditions $u_x(0,t) = 0, u(1,t) = 0.$

Next, using the finite difference method, we start with make a partition for the domain by dividing x-axis into equal steps $h = \frac{b-a}{n} = \frac{1-0}{3} = \frac{1}{3}$, also we dividing t-axis into equal steps $k = 0.05$ as shown in Figure (4.3).

Now, we define the mesh points (x_i, t_j) as follow:

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

$$t_j = c + jk \qquad , j = 0,1,2$$

For $i = 0$, $\qquad x_0 = 0 + 0 \times \frac{1}{3} = 0$

$i = 1$, $\qquad x_1 = 0 + 1 \times \frac{1}{3} = \frac{1}{3}$

$i = 2$, $\qquad x_2 = 0 + 2 \times \frac{1}{3} = \frac{2}{3}$

$i = 3$, $\qquad x_3 = 0 + 3 \times \frac{1}{3} = 1$

And for $j = 0$, $t_0 = 0 + 0 \times 0.05 = 0$

$j = 1$, $t_1 = 0 + 1 \times \frac{1}{3} = 0.05$

$j = 2$, $t_2 = 0 + 2 \times \frac{1}{3} = 0.1$

These partitions will be illustrated in Figure 4.3

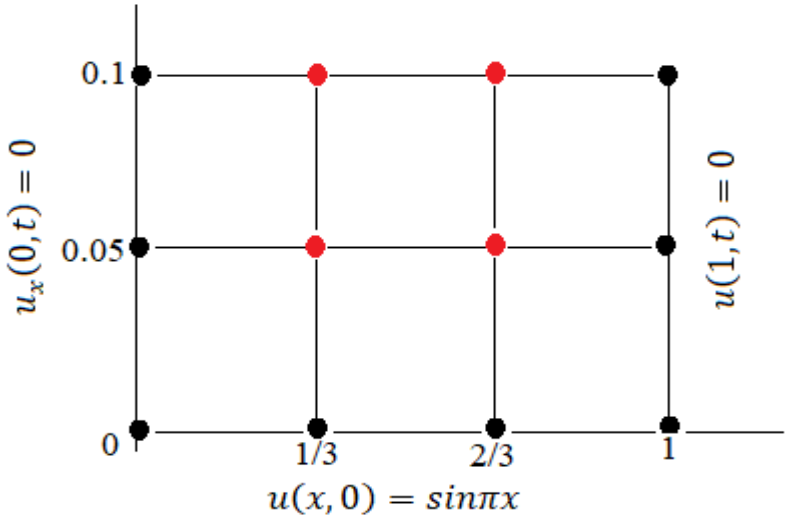

Figure (4.3): Discretization of the domain for example 4. 3

The black points are known boundary points and the red points (interior) points are unknown which are to be approximate.

Using the formula:

$$u_{i,j-1} = (1 + 2\ \lambda)u_{i,j} - \ \lambda\left(u_{i+1,j} + u_{i-1,j}\right)$$

to approximate the interior points.

At the left boundary we will treat the boundary condition by generate ghost boundary x_{-1} by:

$$u_x(x_0, t_n) = \frac{u_{1,j} - u_{-1,j}}{2h} = 0$$

We get

$$u_{1,j} = u_{-1,j} \tag{4.5}$$

Now we back to equation $u_{i,j-1} = (1 + 2\ \lambda)u_{i,j} - \ \lambda\left(u_{i+1,j} + u_{i-1,j}\right)$ and put $i = 0$, yield:

$$u_{0,j-1} = (1 + 2\ \lambda)u_{0,j} - \ \lambda\left(u_{1,j} + u_{-1,j}\right) \tag{4.6}$$

Put equation (4.5) into equation (4.6) we get:

$$u_{0,j-1} = (1 + 2\ \lambda)u_{0,j} - \ 2\lambda\left(u_{1,j}\right) \quad (4.7)$$

The equation (4.7) will used to the left boundary.

We will start to find the interior points:

If $(i = 1, j = 1)$, $u_{1,0} = (1 + 2\ \lambda)u_{1,1} - \ \lambda\left(u_{2,1} + u_{0,1}\right)$

Replace $u_{0,1}$ by using the equation (4.7)

$$u_{0,0} = (1 + 2\ \lambda)u_{0,1} - \ 2\lambda\left(u_{1,1}\right)$$

So $u_{0,1} = \frac{2\lambda}{1+2\ \lambda} u_{1,1}$, put $u_{0,1}$ into equation above yield:

$$u_{1,0} = (1 + 2\ \lambda)u_{1,1} - \ \lambda\left(u_{2,1} + \frac{2\lambda}{1 + 2\ \lambda} u_{1,1}\right)$$

Simplifying this equation yield

$$1.687u_{1,1} - 0.45u_{2,1} = 0.866 \tag{1}$$

If $(i = 2, j = 1), u_{2,0} = (1 + 2\ \lambda)u_{2,1} - \ \lambda\left(u_{3,1} + u_{1,1}\right)$

Also, by some calculation we get the second equation

$$1.9u_{2,1} - 0.45u_{1,1} = 0.5 \qquad (2)$$

If $(i = 1, j = 2), u_{1,1} = (1 + 2\ \lambda)u_{1,2} - \ \lambda\left(u_{2,2} + u_{0,2}\right)$

We do the same argument in first equation to find $u_{0,2}$ and we get third equation

$$1.788u_{1,2} - 0.45u_{2,2} - 1.112u_{1,1} = 0 \qquad (3)$$

If $(i = 2, j = 2), u_{2,1} = (1 + 2\ \lambda)u_{2,2} - \ \lambda\left(u_{3,2} + u_{1,2}\right)$

In the same way we get the fourth equation

$$1.9u_{2,2} - 0.45u_{1,2} - u_{2,1} = 0 \qquad (4)$$

We will use the notation $(u_{1,1} = u_1, u_{1,2} = u_3, u_{2,1} = u_2, u_{2,2} = u_4)$, so equations 1-4 becomes:

$$1.687u_1 - 0.45u_2 = 0.866$$

$$1.9u_2 - 0.45u_1 = 0.5 \qquad (4.8)$$

$$1.788u_3 - 0.45u_4 - 1.112u_1 = 0$$

$$1.9u_4 - 0.45u_3 - u_2 = 0$$

System (4.8) can be expressed as a linear system $Au = b$

$$\begin{bmatrix} 1.687 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1.112 & 0 & 1.788 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

Using $u = A^{-1}b$ we obtain the exact solution

$$u = \begin{bmatrix} 0.6229 \\ 0.4107 \\ 0.4698 \\ 0.3274 \end{bmatrix}$$

We can also solve the linear system (4.8) by the following iterative techniques:

Jacobi Method

We write the Jacobi equations as:

$$\begin{aligned} u_1^{(k)} &= \frac{0.45}{1.687} u_2^{(k-1)} + \frac{0.866}{1.687} \\ u_2^{(k)} &= \frac{0.45}{1.9} u_1^{(k-1)} + \frac{0.5}{1.9} \\ u_3^{(k)} &= \frac{0.45}{1.788} u_4^{(k-1)} + \frac{1.112}{1.788} u_1^{(k-1)} \\ u_4^{(k)} &= \frac{0.45}{1.9} u_3^{(k-1)} + \frac{1}{1.9} u_2^{(k-1)} \end{aligned} \qquad (4.9)$$

Select the initial solution as $u^{(0)} = (0,0,0,0,0,0,0,0)^T$, then we find the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = \frac{0.45}{1.687} u_2^{(0)} + \frac{0.866}{1.687} = \frac{0.866}{1.687} = 0.51333$$

$$u_2^{(1)} = \frac{0.45}{1.9} u_1^{(0)} + \frac{0.5}{1.9} = \frac{0.5}{1.9} = 0.26315$$

$$u_3^{(1)} = \frac{0.45}{1.788}u_4^{(0)} + \frac{1.112}{1.788}u_1^{(0)} = 0$$

$$u_4^{(1)} = \frac{0.45}{1.9}u_3^{(0)} + \frac{1}{1.9}u_2^{(0)} = 0$$

The first iteration $u^{(1)}$ is:

$$u^{(1)} = \begin{bmatrix} 0.51333 \\ 0.26315 \\ 0 \\ 0 \end{bmatrix}$$

Likewise, after 15 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.622885 \\ 0.410683 \\ 0.469790 \\ 0.327415 \end{bmatrix}$$

Number of iterations	The error
15	$5.254187 \times 10^{-008}$

The Matlab code for the Jacobi iterative method can be formed in Appendix E.

Gauss-Seidel Method

We write the Gauss-Seidel equations as:

$$u_1^{(k)} = \frac{0.45}{1.687}u_2^{(k-1)} + \frac{0.866}{1.687}$$

$$u_2^{(k)} = \frac{0.45}{1.9}u_1^{(k)} + \frac{0.5}{1.9}$$

$$u_3^{(k)} = \frac{0.45}{1.788}u_4^{(k-1)} + \frac{1.112}{1.788}u_1^{(k)} \qquad (4.10)$$

$$u_4^{(k)} = \frac{0.45}{1.9}u_3^{(k)} + \frac{1}{1.9}u_2^{(k)}$$

Choose the initial solution as $u^{(0)} = (0,0,0,0,)^T$, then we find the first iteration $u^{(1)}$as:

$$u_1^{(1)} = \frac{0.45}{1.687}u_2^{(0)} + \frac{0.866}{1.687} = \frac{0.866}{1.687} = 0.513337$$

$$u_2^{(1)} = \frac{0.45}{1.9}u_1^{(1)} + \frac{0.5}{1.9} = \frac{0.45 \times 0.513337}{1.9} + \frac{0.5}{1.9} = 0.384737$$

$$u_3^{(1)} = \frac{0.45}{1.788}u_4^{(0)} + \frac{1.112}{1.788}u_1^{(1)} = \frac{1.112 \times 0.513337}{1.788} = 0.319256$$

$$u_4^{(1)} = \frac{0.45}{1.9}u_3^{(1)} + \frac{1}{1.9}u_2^{(1)} = \frac{0.45 \times 0.319256}{1.9} + \frac{0.384737}{1.9} = 0.278106$$

The first iteration is

$$u^{(1)} = \begin{bmatrix} 0.513337 \\ 0.384737 \\ 0.319256 \\ 0.278106 \end{bmatrix}$$

Likewise, after 8 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.622885 \\ 0.410683 \\ 0.469790 \\ 0.327415 \end{bmatrix}$$

Number of iterations	The error
8	$4.405655 \times 10^{-008}$

The Matlab code for the Gauss-Seidel iterative method can be formed in Appendix F.

Successive Over Relaxation (SOR) Method

Writing the SOR equations by using Gauss-Seidel equations and use the relaxation factor $\omega = 1.3$ we get

$$u_1^{(k)} = (1-1.3)u_1^{(k-1)} + 1.3\left[\frac{0.45}{1.687}u_2^{(k-1)} + \frac{0.866}{1.687}\right]$$

$$u_2^{(k)} = (1-1.3)u_2^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_1^{(k)} + \frac{0.5}{1.9}\right] \qquad (4.11)$$

$$u_3^{(k)} = (1-1.3)u_3^{(k-1)} + 1.3\left[\frac{0.45}{1.788}u_4^{(k-1)} + \frac{1.112}{1.788}u_1^{(k)}\right]$$

$$u_4^{(k)} = (1-1.3)u_4^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_3^{(k)} + \frac{1}{1.9}u_2^{(k)}\right]$$

Select the initial solution as $u^{(0)} = (0,0,0,0)^T$, then we find the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = (1-1.3)u_1^{(0)} + 1.3\left[\frac{0.45}{1.687}u_2^{(0)} + \frac{0.866}{1.687}\right] = \frac{1.3 \times 0.866}{1.687}$$
$$= 0.667338$$

$$u_2^{(1)} = (1-1.3)u_2^{(0)} + 1.3\left[\frac{0.45}{1.9}u_1^{(1)} + \frac{0.5}{1.9}\right]$$
$$= 1.3\left[\frac{0.45 \times 0.667338}{1.9} + \frac{0.5}{1.9}\right] = 0.547575$$

$$u_3^{(1)} = (1-1.3)u_3^{(0)} + 1.3\left[\frac{0.45}{1.788}u_4^{(0)} + \frac{1.112}{1.788}u_1^{(1)}\right]$$
$$= 1.3\left[\frac{1.112 \times 0.667338}{1.788}\right] = 0.539543$$

$$u_4^{(1)} = (1-1.3)u_4^{(0)} + 1.3\left[\frac{0.45}{1.9}u_3^{(1)} + \frac{1}{1.9}u_2^{(1)}\right]$$
$$= 1.3\left[\frac{0.45 \times 0.539543}{1.9} + \frac{0.547575}{1.9}\right] = 0.540779$$

The first iteration $u^{(1)}$ gives

$$u^{(1)} = \begin{bmatrix} 0.667338 \\ 0.547575 \\ 0.539543 \\ 0.540779 \end{bmatrix}$$

Likewise, after 6 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.622885 \\ 0.410683 \\ 0.469790 \\ 0.327415 \end{bmatrix}$$

Number of iterations	The error
6	$8.037576 \times 10^{-008}$

The Matlab code for the SOR iterative method can be formed in Appendix G.

Conjugate Gradient Method

This algorithm can be implemented as follows:

Step1 Start with initial guess $u_0 = (0,0,0,0)^T$

Step2 Calculate the residual vector r_0 as follows:

$$r_0 = b - Au_0$$

$$r_0 = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1.687 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1.112 & 0 & 1.788 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$r_0 = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

Step 3: Let the initial direction vector $p_0 = r_0$. So

$$p_0 = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

Step 4 compute the scalar $\alpha_k's$ by formula

$$\alpha_k = \frac{r_k^t r_k}{p_k^t A p_k}$$

for $k = 0$,

$$\alpha_0 = \frac{r_0^t r_0}{p_0^t A p_0}$$

$$\mathrm{r}_0^{\mathrm{t}}\mathrm{r}_0 = [0.866 \quad 0.5 \quad 0 \quad 0] \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} = 0.999956$$

$$\mathrm{p}_0^{\mathrm{t}}\mathrm{Ap}_0 = [0.866 \quad 0.5 \quad 0 \quad 0] \begin{bmatrix} 1.687 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1.112 & 0 & 1.788 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

$$= 1.350475$$

Thus,

$$\alpha_0 = \frac{0.999956}{1.350475} = 0.740447$$

Step5 Compute the first iteration u_1 by the formula

$$u_1 = u_0 + \alpha_0 p_0$$

$u_1 = (0,0,0,0) + 0.740447(0.866,0.5,0,0)$

the first iteration $u_1 = \begin{bmatrix} 0.641227 \\ 0.370223 \\ 0 \\ 0 \end{bmatrix}$

Likewise, after 5 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.622885 \\ 0.410683 \\ 0.469790 \\ 0.327415 \end{bmatrix}$$

Number of iterations	The error
5	2.52240×10^{-009}

The Matlab code for the conjugate gradient iterative method can be formed in Appendix H.

Table (4.4): Comparison between iterative methods in example 4.3

Methods / u	Jacobi Method	Gauss-Seidel Method	SOR Method	Conjugate Gradient
u_1	0.622885	0.622885	0.622885	0.622885
u_2	0.410683	0.410683	0.410683	0.410683
u_3	0.469790	0.469790	0.469790	0.469790
u_4	0.327415	0.327415	0.327415	0.327415
Number of iterations	15	8	6	5
Error	5.2541×10^{-008}	4.4056×10^{-008}	8.0375×10^{-008}	2.52240×10^{-009}

Example 4.4: Consider the one-dimensional heat equation

$$u_t = u_{xx}, \qquad 0 \le x \le 1, t \ge 0$$

Subject to initial condition $u(x,0) = sin\pi x$ and boundary conditions

$u_x(0,t) - u(0,t) = 0, u(1,t) = 0.$

Next, using the finite difference method, we start with make a partition for the domain by dividing x-axis into equal steps $h = \frac{b-a}{n} = \frac{1-0}{3} = \frac{1}{3}$, also we dividing t-axis into equal steps $k = 0.05$ as shown in Figure (4.4).

Now, we define the mesh points (x_i, t_j) as follow:

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

$$t_j = c + jk \quad , j = 0,1,2$$

For $i = 0$, $x_0 = 0 + 0 \times \frac{1}{3} = 0$

$i = 1$, $x_1 = 0 + 1 \times \frac{1}{3} = \frac{1}{3}$

$i = 2$, $x_2 = 0 + 2 \times \frac{1}{3} = \frac{2}{3}$

$i = 3$, $x_3 = 0 + 3 \times \frac{1}{3} = 1$

And for $j = 0$, $t_0 = 0 + 0 \times 0.05 = 0$

$j = 1$, $t_1 = 0 + 1 \times \frac{1}{3} = 0.05$

$j = 2$, $t_2 = 0 + 2 \times \frac{1}{3} = 0.1$

These partitions will be illustrated in Figure 4.3

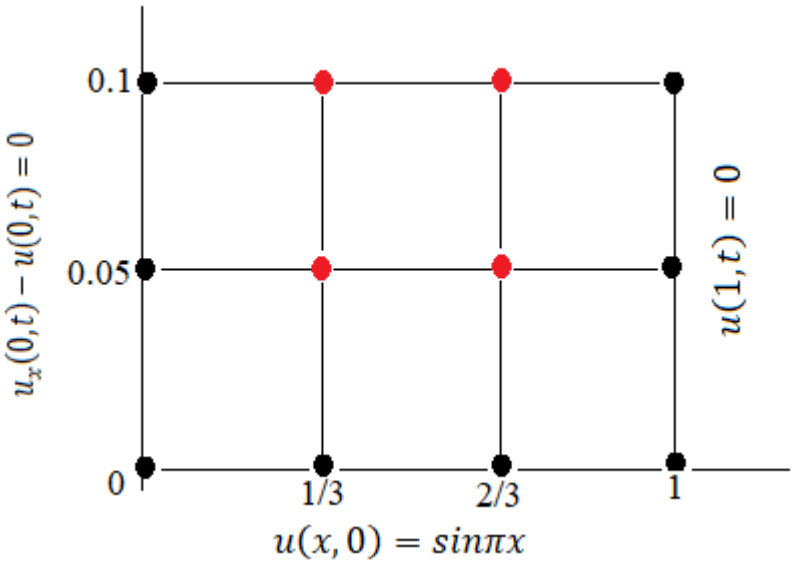

Figure (4.4): Discretization of the domain for example 4. 4

The black points are known boundary points and the red points (interior) points are unknown which are to be approximate.

To approximate the interior points, we use the formula:

$$u_{i,j-1} = (1 + 2\ \lambda)u_{i,j} - \ \lambda\left(u_{i+1,j} + u_{i-1,j}\right)$$

At the left boundary we will treat the boundary condition by generate ghost boundary x_{-1} by:

$$u_x(x_0, t_n) = \frac{u_{1,j} - u_{-1,j}}{2h} = u(0, t_j)$$

We get

$$u_{-1,j} = u_{1,j} - 2hu(0, t_j) \qquad (4.12)$$

Now we back to equation $u_{i,j-1} = (1 + 2\ \lambda)u_{i,j} - \ \lambda\left(u_{i+1,j} + u_{i-1,j}\right)$ and put $i = 0$, yield:

$$u_{0,j-1} = (1 + 2\ \lambda)u_{0,j} - \ \lambda\left(u_{1,j} + u_{-1,j}\right) \qquad (4.13)$$

Putting equation (4.12) into equation (4.13) we get:

$$u_{0,j-1} = (1 + 2\ \lambda)u_{0,j} - \ 2\lambda\left(u_{1,j} - hu(0,t_j)\right) \qquad (4.14)$$

Equation (4.14) will used to the left boundary.

If $(i = 1, j = 1)$, $u_{1,0} = (1 + 2\ \lambda)u_{1,1} - \ \lambda\left(u_{2,1} + u_{0,1}\right)$

Replace $u_{0,1}$ by using equation (4.14)

$$u_{0,0} = (1 + 2\ \lambda)u_{0,1} - \ 2\lambda\left(u_{1,1} - hu(0,t_1\right)$$

So $u_{0,1} = \frac{2\lambda}{1+2\ \lambda\ +2\ \lambda\,\mathrm{h}}u_{1,1}$, put $u_{0,1}$ into equation above yield:

$$u_{1,0} = (1 + 2\ \lambda)u_{1,1} - \ \lambda\left(u_{2,1} + \frac{2\lambda}{1 + 2\ \lambda\ + 2\ \lambda\,\mathrm{h}}u_{1,1}\right)$$

Then we get fist equation

$$1.4909u_{1,1} - 0.45u_{2,1} = 0.866 \qquad (1)$$

If $(i = 2, j = 1)$, $u_{2,0} = (1 + 2\ \lambda)u_{2,1} - \ \lambda\left(u_{3,1} + u_{1,1}\right)$

Also, we get second equation

$$1.9u_{2,1} - 0.45u_{1,1} = 0.5 \qquad (2)$$

If $(i = 1, j = 2)$, $u_{1,1} = (1 + 2\ \lambda)u_{1,2} - \ \lambda\left(u_{2,2} + u_{0,2}\right)$

We do the same argument in first equation to find $u_{0,2}$, then we get third equation.

$$1.7159u_{1,2} - 0.45u_{2,2} - 1.08367u_{1,1} = 0 \qquad (3)$$

If $(i = 2, j = 2)$, $u_{2,1} = (1 + 2\ \lambda)u_{2,2} - \ \lambda\left(u_{3,2} + u_{1,2}\right)$

In the same way we get the fourth equation

$$1.9u_{2,2} - 0.45u_{1,2} - u_{2,1} = 0 \qquad (4)$$

We will use the notation $(u_{1,1} = u_1, u_{1,2} = u_3, u_{2,1} = u_2, u_{2,2} = u_4)$, then equations 1-4 becomes:

$$1.4909u_1 - 0.45u_2 = 0.866$$

$$1.9u_2 - 0.45u_1 = 0.5 \qquad (4.15)$$

$$1.7159u_3 - 0.45u_4 - 1.08367u_1 = 0$$

$$1.9u_4 - 0.45u_3 - u_2 = 0$$

System (4.15) can be expressed in a matrix form as follows:

$$\begin{bmatrix} 1.4909 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1.08367 & 0 & 1.7159 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

Using $u = A^{-1}b$ we obtain the exact solution

$$u = \begin{bmatrix} 0.7111 \\ 0.4316 \\ 0.5424 \\ 0.3556 \end{bmatrix}$$

We can also solve the linear system (4.15) by the following iterative techniques:

Jacobi Method

We write the Jacobi equations as:

$$u_1{}^{(k)} = \frac{0.45}{1.4909} u_2{}^{(k-1)} + \frac{0.866}{1.4909}$$

$$u_2^{(k)} = \frac{0.45}{1.9} u_1^{(k-1)} + \frac{0.5}{1.9} \qquad (4.16)$$

$$u_3^{(k)} = \frac{0.45}{1.7159} u_4^{(k-1)} + \frac{1.08367}{1.7159} u_1^{(k-1)}$$

$$u_4^{(k)} = \frac{0.45}{1.9} u_3^{(k-1)} + \frac{1}{1.9} u_2^{(k-1)}$$

Choose the initial solution as $u^{(0)} = (0,0,0,0)^T$, then we find the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = \frac{0.45}{1.4909} u_2^{(0)} + \frac{0.866}{1.4909} = 0.580857$$

$$u_2^{(1)} = \frac{0.45}{1.9} u_1^{(0)} + \frac{0.5}{1.9} = 0.263\,157$$

$$u_3^{(1)} = \frac{0.45}{1.7159} u_4^{(0)} + \frac{1.08367}{1.7159} u_1^{(0)} = 0$$

$$u_4^{(1)} = \frac{0.45}{1.9} u_3{}^0 u_3^{(0)} + \frac{1}{1.9} u_2^{(0)} = 0$$

The first iteration $u^{(1)}$ is:

$$u^{(1)} = \begin{bmatrix} 0.580857 \\ 0.263157 \\ 0 \\ 0 \end{bmatrix}$$

Likewise, after 16 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.711121 \\ 0.43158 \\ 0.542364 \\ 0.355602 \end{bmatrix}$$

Number of iterations	The error
16	3.02119×10^{-008}

The Matlab code for the Jacobi iterative method can be formed in Appendix I.

Gauss-Seidel Method

We start with writing the Gauss-Seidel equations as:

$$u_1^{(k)} = \frac{0.45}{1.4909} u_2^{(k-1)} + \frac{0.866}{1.4909}$$

$$u_2^{(k)} = \frac{0.45}{1.9} u_1^{(k)} + \frac{0.5}{1.9} \qquad (4.17)$$

$$u_3^{(k)} = \frac{0.45}{1.7159} u_4^{(k-1)} + \frac{1.08367}{1.7159} u_1^{(k)}$$

$$u_4^{(k)} = \frac{0.45}{1.9} u_3^{(k)} + \frac{1}{1.9} u_2^{(k)}$$

Choose the initial solution as $u^{(0)} = (0,0,0,0,)^T$, then we obtain the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = \frac{0.45}{1.4909} u_2^{(0)} + \frac{0.866}{1.4909} = \frac{0.866}{1.4909} = 0.580857$$

$$u_2^{(1)} = \frac{0.45}{1.9} u_1^{(1)} + \frac{0.5}{1.9} = \frac{0.45 \times 0.580857}{1.9} + \frac{0.5}{1.9} = 0.400729$$

$$u_3^{(1)} = \frac{0.45}{1.7159} u_4^{(0)} + \frac{1.08367}{1.7159} u_1^{(1)} = \frac{1.08367 \times 0.580857}{1.7159} = 0.366838$$

$$u_4^{(1)} = \frac{0.45}{1.9}u_3^{(1)} + \frac{1}{1.9}u_2^{(1)} = \frac{0.45 \times 0.366838}{1.9} + \frac{0.400729}{1.9}$$

$$= 0.297792$$

The first iteration is

$$u^{(1)} = \begin{bmatrix} 0.580857 \\ 0.400729 \\ 0.366838 \\ 0.297792 \end{bmatrix}$$

Likewise, after 14 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.711121 \\ 0.431581 \\ 0.542364 \\ 0.355602 \end{bmatrix}$$

Number of iterations	The error
14	$2.512671 \times 10^{-008}$

The Matlab code for the Gauss-Seidel iterative method can be formed in Appendix J.

Successive Over Relaxation (SOR) Method

We start with writing the SOR equations by using Gauss-Seidel equations by using relaxation factor $\omega = 1.3$ we get

$$u_1^{(k)} = (1 - 1.3)u_1^{(k-1)} + 1.3\left[\frac{0.45}{1.4909}u_2^{(k-1)} + \frac{0.866}{1.4909}\right]$$

$$u_2^{(k)} = (1 - 1.3)u_2^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_1^{(k)} + \frac{0.5}{1.9}\right] \qquad (4.18)$$

$$u_3^{(k)} = (1 - 1.3)u_3^{(k-1)} + 1.3\left[\frac{0.45}{1.7159}u_4^{(k-1)} + \frac{1.08367}{1.7159}u_1^{(k)}\right]$$

$$u_4^{(k)} = (1 - 1.3)u_4^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_3^{(k)} + \frac{1}{1.9}u_2^{(k)}\right]$$

Choose the initial solution as $u^{(0)} = (0,0,0,0)^T$, then we obtain the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = (1-1.3)u_1^{(0)} + 1.3\left[\frac{0.45}{1.4909}u_2^{(2)} + \frac{0.866}{1.4909}\right] = \frac{1.3 \times 0.866}{1.4909}$$
$$= 0.755114$$

$$u_2^{(1)} = (1-1.3)u_2^{(0)} + 1.3\left[\frac{0.45}{1.9}u_1^{(1)} + \frac{0.5}{1.9}\right]$$
$$= 1.3\left[\frac{0.45 \times 0.755114}{1.9} + \frac{0.5}{1.9}\right] = 0.57460$$

$$u_3^{(1)} = (1-1.3)u_3^{(0)} + 1.3\left[\frac{0.45}{1.7159}u_4^{(0)} + \frac{1.08367}{1.7159}u_1^{(1)}\right]$$
$$= 1.3\left[\frac{1.08367 \times 0.755114}{1.7159}\right] = 0.619956$$

$$u_4^{(1)} = (1-1.3)u_4^{(0)} + 1.3\left[\frac{0.45}{1.9}u_3^{(1)} + \frac{1}{1.9}u_2^{(1)}\right]$$
$$= 1.3\left[\frac{0.45 \times 0.619956}{1.9} + \frac{10.574601}{1.9}\right] = 0.584029$$

The first iteration $u^{(1)}$ is:

$$u^1 = \begin{bmatrix} 0.755114 \\ 0.574601 \\ 0.619956 \\ 0.584029 \end{bmatrix}$$

Likewise, after 11 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.711121 \\ 0.431581 \\ 0.542364 \\ 0.355602 \end{bmatrix}$$

Number of iterations	The error
11	$2.223857 \times 10^{-008}$

The Matlab code for the SOR iterative method can be formed in Appendix K.

Conjugate Gradient Method

This algorithm can be implemented as follows:

Step1 Start with initial guess $u_0 = (0,0,0,0)^T$

Step2 Calculate the residual vector r_0 as follows:

$$r_0 = b - Au_0$$

$$r_0 = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1.4909 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1.08367 & 0 & 1.7159 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$r_0 = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

Step 3: Let the initial direction vector $p_0 = r_0$. So

$$p_0 = \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

Step 4 compute the scalar $\alpha_k's$ by formula

$$\alpha_k = \frac{r_k^t r_k}{p_k^t A p_k}$$

for $k = 0$,

$$\alpha_0 = \frac{r_0^t r_0}{p_0^t A p_0}$$

$$\mathrm{r}_0^{\mathrm{t}}\mathrm{r}_0 = [0.866 \quad 0.5 \quad 0 \quad 0] \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} = 0.999956$$

$$p_0^t A p_0 = [0.866 \quad 0.5 \quad 0 \quad 0] \begin{bmatrix} 1.4909 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1.08367 & 0 & 1.7159 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} 0.866 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}$$

$$= 1.203409$$

Thus

$$\alpha_0 = \frac{0.999956}{1.203409} = 0.830936$$

Step5 Compute the first iteration u_1 by the formula

$$u_1 = u_0 + \alpha_0 p_0$$

$$u_1 = (0,0,0,0) + 0.830936(0.866,0.5,0,0)$$

The first iteration is:

$$u_1 = \begin{bmatrix} 0.719590 \\ 0.415468 \\ 0 \\ 0 \end{bmatrix}$$

Likewise, after 5 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.711121 \\ 0.431581 \\ 0.542364 \\ 0.355602 \end{bmatrix}$$

Number of iterations	The error
5	5.68523×10^{-009}

The Matlab code for the conjugate gradient iterative method can be formed in Appendix L.

Table (4.5): Comparison between iterative methods in example 4.4

Methods / u	Jacobi Method	Gauss-Seidel Method	SOR Method	Conjugate Gradient
u_1	0.711121	0.711121	0.711121	0.711121
u_2	0.431581	0.431581	0.431581	0.431581
u_3	0.542364	0.542364	0.542364	0.542364
u_4	0.355602	0.355602	0.355602	0.355602
Number of iterations	16	14	11	5
Error	3.0119×10^{-008}	$2.512671 \times 10^{-008}$	$2.223857 \times 10^{-008}$	$5.685235 \times 10^{-009}$

Example 4.5: Consider the one-dimensional heat equation

$$u_t = u_{xx} + xt, \qquad 0 \le x \le 1, t \ge 0$$

subject to initial condition $u(x,0) = sin\pi x$ and boundary conditions $u(0,t) = 0, u(1,t) = 0.$

Next, using finite difference method, we start with make a partition for the domain by dividing x-axis into equal steps$h = \frac{b-a}{n} = \frac{1-0}{3} = \frac{1}{3}$, also we dividing t-axis into equal steps $k = 0.05$ as shown in Figure (4.5).

We define the mesh points (x_i, t_j) as follow:

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

$$t_j = c + jk \quad , j = 0,1,2$$

For $i = 0$, $x_0 = 0 + 0 \times \frac{1}{3} = 0$

$i = 1$, $x_1 = 0 + 1 \times \frac{1}{3} = \frac{1}{3}$

$i = 2$, $x_2 = 0 + 2 \times \frac{1}{3} = \frac{2}{3}$

$i = 3$, $x_3 = 0 + 3 \times \frac{1}{3} = 1$

And for $j = 0$, $t_0 = 0 + 0 \times 0.05 = 0$

$j = 1$, $t_1 = 0 + 1 \times \frac{1}{3} = 0.05$

$j = 2$, $t_2 = 0 + 2 \times \frac{1}{3} = 0.1$

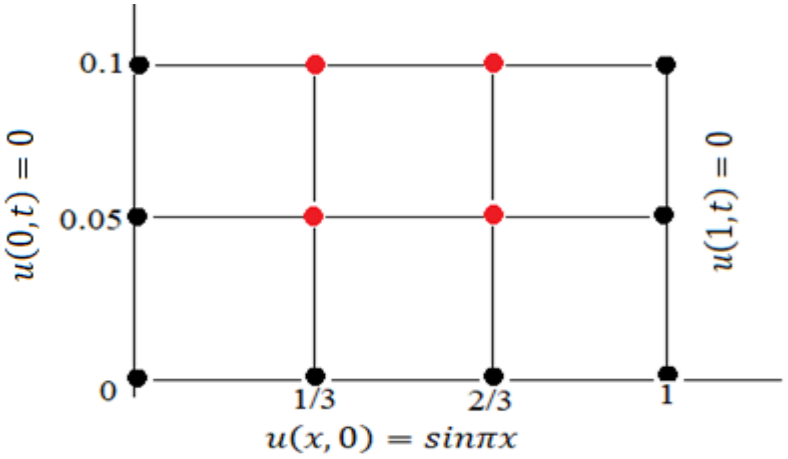

Figure (4.5): Discretization of the domain for example 4. 5

The black points are known boundary points and the red points (interior) points are unknown which are to be approximate.

To approximate the interior points, we use the formula:

$$u_{i,j-1} = u_{i,j} - k\, h(x_i, t_j) + \frac{\alpha^2 k}{h^2}\left(2u_{i,j} - u_{i-1,j} - u_{i+1,j}\right)$$

We will start to find the interior points:

If $(i = 1, j = 1)$, $u_{1,0} = u_{1,1} - k\, h(x_1, t_1) + \frac{\alpha^2 k}{h^2}\left(2u_{1,1} - u_{0,1} - u_{2,1}\right)$

Inserting the boundary conditions in the previous equation yield

$$1.9u_{1,1} - 0.45u_{2,1} = 0.86683 \qquad (1)$$

If $(i = 2, j = 1)$, $u_{2,0} = u_{2,1} - k\ h(x_2, t_1) + \frac{\alpha^2 k}{h^2}\left(2u_{2,1} - u_{1,1} - u_{3,1}\right)$

Inserting the boundary conditions in the previous equation yield

$$1.9u_{2,1} - 0.45u_{1,1} = 0.50167 \qquad (2)$$

If$(i = 1, j = 2)$,$u_{1,1} = u_{1,2} - k\ h(x_1, t_2) + \frac{\alpha^2 k}{h^2}\left(2u_{1,2} - u_{0,2} - u_{2,2}\right)$

Inserting the boundary conditions in the previous equation yield

$$1.9u_{1,2} - 0.45u_{2,2} - u_{1,1} = 1.666 \times 10^{-3} \qquad (3)$$

If $(i = 2, j = 2)$, $u_{2,1} = u_{2,2} - k\ h(x_2, t_2) + \frac{\alpha^2 k}{h^2}\left(2u_{2,2} - u_{1,2} - u_{3,2}\right)$

Inserting the boundary conditions in the previous equation yield

$$1.9u_{2,2} - 0.45u_{1,2} - u_{2,1} = 3.333 \times 10^{-3} \qquad (4)$$

We will use the notation $(u_{1,1} = u_1, u_{2,1} = u_2, u_{1,2} = u_3, u_{2,2} = u_4)$, then equations 1-4 becomes:

$$1.9u_1 - 0.45u_2 = 0.86683$$

$$1.9u_2 - 0.45u_1 = 0.50167$$

$$1.9u_3 - 0.45u_4 - u_1 = 1.666 \times 10^{-3} \qquad (4.19)$$

$$1.9u_4 - 0.45u_3 - u_2 = 3.333 \times 10^{-3}$$

System (4.19) can be expressed in matrix form as follows:

$$\begin{bmatrix} 1.9 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1 & 0 & 1.9 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.86683 \\ 0.50167 \\ 1.666 \times 10^{-3} \\ 3.333 \times 10^{-3} \end{bmatrix}$$

Using $u = A^{-1}b$ we obtain the exact solution:

$$u = \begin{bmatrix} 0.5496 \\ 0.3942 \\ 0.3599 \\ 0.2945 \end{bmatrix}$$

We can also solve the linear system (4.19) by the following iterative techniques:

Jacobi Method

We write the Jacobi equations as:

$$\begin{aligned} u_1^{(k)} &= \frac{0.45}{1.9} u_2^{(k-1)} + \frac{0.86683}{1.9} \\ u_2^{(k)} &= \frac{0.45}{1.9} u_1^{(k-1)} + \frac{0.50167}{1.9} \\ u_3^{(k)} &= \frac{0.45}{1.9} u_4^{(k-1)} + \frac{1}{1.9} u_1^{(k-1)} + \frac{1.6667 \times 10^{-3}}{1.9} \\ u_4^{(k)} &= \frac{0.45}{1.9} u_3^{(k-1)} + \frac{1}{1.9} u_2^{(k-1)} + \frac{3.333 \times 10^{-3}}{1.9} \end{aligned} \qquad (4.20)$$

Choose the initial solution as $u^{(0)} = (0,0,0,0)^T$, then we find the first iteration $u^{(1)}$ as:

$$\begin{aligned} u_1^{(1)} &= \frac{0.45}{1.9} u_2^{(0)} + \frac{0.86683}{1.9} = 0.45622 \\ u_2^{(1)} &= \frac{0.45}{1.9} u_1^{(0)} + \frac{0.50167}{1.9} = 0.26403 \\ u_3^{(1)} &= \frac{0.45}{1.9} u_4^{(0)} + \frac{1}{1.9} u_1^{(0)} + \frac{1.6667 \times 10^{-3}}{1.9} = 8.7721 \times 10^{-3} \\ u_4^{(1)} &= \frac{0.45}{1.9} u_3^{(0)} + \frac{1}{1.9} u_2^{(0)} + \frac{3.333 \times 10^{-3}}{1.9} = 1.75421 \times 10^{-3} \end{aligned}$$

The first iteration $u^{(1)}$ is:

$$u^{(1)} = \begin{bmatrix} 0.45622 \\ 0.26403 \\ 08.7721 \times 10^{-3} \\ 1.75421 \times 10^{-3} \end{bmatrix}$$

Likewise, after 14 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.54959 \\ 0.39420 \\ 0.35987 \\ 0.29446 \end{bmatrix}$$

Number of iterations	The error
14	9.72601×10^{-008}

The Matlab code for the Jacobi iterative method can be formed in Appendix E.

Gauss-Seidel Method

We start with writing the Gauss-Seidel equations as:

$$\begin{aligned} u_1^{(k)} &= \frac{0.45}{1.9} u_2^{(k-1)} + \frac{0.86683}{1.9} \\ u_2^{(k)} &= \frac{0.45}{1.9} u_1^{(k)} + \frac{0.50167}{1.9} \\ u_3^{(k)} &= \frac{0.45}{1.9} u_4^{(k-1)} + \frac{1}{1.9} u_1^{(k)} + \frac{1.6667 \times 10^{-3}}{1.9} \\ u_4^{(k)} &= \frac{0.45}{1.9} u_3^{(k)} + \frac{1}{1.9} u_2^{(k)} + \frac{3.333 \times 10^{-3}}{1.9} \end{aligned} \qquad (4.21)$$

Choose the initial solution as $u^{(0)} = (0,0,0,0,)^T$, then we find the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = \frac{0.45}{1.9}u_2^{(0)} + \frac{0.86683}{1.9} = 0.45622$$

$$u_2^{(1)} = \frac{0.45}{1.9}u_1^{(1)} + \frac{0.50167}{1.9} = \frac{0.45 \times 0.45622}{1.9} + \frac{0.50167}{1.9} = 0.37208$$

$$u_3^{(1)} = \frac{0.45}{1.9}u_4^{(0)} + \frac{1}{1.9}u_1^{(1)} + \frac{1.6667 \times 10^{-3}}{1.9} = 0.24099$$

$$u_4^{(1)} = \frac{0.45}{1.9}u_3^{(1)} + \frac{1}{1.9}u_2^{(1)} + \frac{3.333 \times 10^{-3}}{1.9} = 0.25$$

The first iteration is:

$$u^{(1)} = \begin{bmatrix} 0.45622 \\ 0.37208 \\ 0.24099 \\ 0.25466 \end{bmatrix}$$

Likewise, after 8 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.54959 \\ 0.39420 \\ 0.35987 \\ 0.29446 \end{bmatrix}$$

Number of iterations	The error
8	2.06637×10^{-008}

The Matlab code for the Gauss-Seidel iterative method can be formed in Appendix F.

Successive Over Relaxation (SOR) Method

We will start with writing the SOR equations by using Gauss-Seidel equations and use the relaxation factor $\omega = 1.3$. We get:

$$u_1^{(k)} = (1 - 1.3)u_1^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_2^{(k-1)} + \frac{0.86683}{1.9}\right]$$

$$u_2^{(k)} = (1-1.3)u_2^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_1^{(k)} + \frac{0.50167}{1.9}\right] \quad (4.22)$$

$$u_3^{(k)} = (1-1.3)u_3^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_4^{(k-1)} + \frac{1}{1.9}u_1^{(k)} + \frac{1.6667\times10^{-3}}{1.9}\right]$$

$$u_4^{(k)} = (1-1.3)u_4^{(k-1)} + 1.3\left[\frac{0.45}{1.9}u_3^{(k)} + \frac{1}{1.9}u_2^{(k)} + \frac{3.333\times10^{-3}}{1.9}\right]$$

Choose the initial solution as $u^{(0)} = (0,0,0,0)^T$, then we find the first iteration $u^{(1)}$ as:

$$u_1^{(1)} = (1-1.3)u_1^{(0)} + 1.3\left[\frac{0.45}{1.9}u_2^{(0)} + \frac{0.86683}{1.9}\right] = 0.46535$$

$$u_2^{(1)} = (1-1.3)u_2^{(0)} + 1.3\left[\frac{0.45}{1.9}u_1^{(1)} + \frac{0.50167}{1.9}\right] = 0.381735$$

$$u_3^{(1)} = (1-1.3)u_3^{(0)} + 1.3\left[\frac{0.45}{1.9}u_4^{(0)} + \frac{1}{1.9}u_1^{(1)} + \frac{1.6667\times10^{-3}}{1.9}\right]$$

$$= 0.25071$$

$$u_4^{(1)} = (1-1.3)u_4^{(0)} + 1.3\left[\frac{0.45}{1.9}u_3^{(1)} + \frac{1}{1.9}u_2^{(1)} + \frac{3.333\times10^{-3}}{1.9}\right]$$

$$= 0.26728$$

The first iteration $u^{(1)}$ is:

$$u^{(1)} = \begin{bmatrix} 0.46535 \\ 0.38173 \\ 0.25071 \\ 0.26728 \end{bmatrix}$$

Likewise, after 6 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.54959 \\ 0.39420 \\ 0.35987 \\ 0.29446 \end{bmatrix}$$

Number of iterations	The error
6	8.32123×10^{-008}

The Matlab code for the SOR iterative method can be formed in Appendix G.

Conjugate Gradient Method

This algorithm can be implemented as follows:

Step1 Start with initial guess $u_0 = (0,0,0,0)^T$

Step2 Calculate the residual vector r_0 as follows:

$$r_0 = b - Au_0$$

$$r_0 = \begin{bmatrix} 0.86683 \\ 0.50167 \\ 1.666 \times 10^{-3} \\ 3.333 \times 10^{-3} \end{bmatrix} - \begin{bmatrix} 1.9 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1 & 0 & 1.9 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$r_0 = \begin{bmatrix} 0.86683 \\ 0.50167 \\ 1.666 \times 10^{-3} \\ 3.333 \times 10^{-3} \end{bmatrix}$$

Step 3: Let the initial direction vector $p_0 = r_0$. So

$$p_0 = \begin{bmatrix} 0.86683 \\ 0.50167 \\ 1.666 \times 10^{-3} \\ 3.333 \times 10^{-3} \end{bmatrix}$$

Step 4 compute the scalar $\alpha_k's$ by formula

$$\alpha_k = \frac{r_k^t r_k}{p_k^t A p_k}$$

for $k = 0$,

$$\alpha_k = \frac{r_0^t r_0}{p_0^t A p_0}$$

$$\mathrm{r}_0^{\mathrm{t}}\mathrm{r}_0 = \begin{bmatrix} 0.86683 & 0.50167 & \frac{1.666}{1000} & \frac{3.333}{1000} \end{bmatrix} \begin{bmatrix} 0.86683 \\ 0.50167 \\ \frac{1.666}{1000} \\ \frac{3.333}{1000} \end{bmatrix} = 1.00308$$

$\mathrm{p}_0^{\mathrm{t}}\mathrm{A}\mathrm{p}_0$

$$= \begin{bmatrix} 0.86683 & 0.50167 & \frac{1.666}{1000} & \frac{3.333}{1000} \end{bmatrix} \begin{bmatrix} 1.9 & -0.45 & 0 & 0 \\ -0.45 & 1.9 & 0 & 0 \\ -1 & 0 & 1.9 & -0.45 \\ 0 & -1 & -0.45 & 1.9 \end{bmatrix} \begin{bmatrix} 0.86683 \\ 0.50167 \\ \frac{1.666}{1000} \\ \frac{3.333}{1000} \end{bmatrix}$$

$= 1.51135$

Thus

$$\alpha_0 = \frac{1.00308}{1.51135} = 0.66369$$

Step5 Compute the first iteration u_1 by the formula

$$u_1 = u_0 + \alpha_0 p_0$$

$$u_1 = (0,0,0,0) + 0.66369 \begin{bmatrix} 0.86683 \\ 0.50167 \\ 1.666 \times 10^{-3} \\ 3.333 \times 10^{-3} \end{bmatrix}$$

The first iteration is

$$u_1 = \begin{bmatrix} 0.57530 \\ 0.33295 \\ 0.00110 \\ 0.00221 \end{bmatrix}$$

Likewise, after 5 iterations we obtain the approximate solution:

$$u = \begin{bmatrix} 0.54959 \\ 0.3943 \\ 0.35971 \\ 0.29455 \end{bmatrix}$$

Number of iterations	The error
5	$13.872124 \times 10^{-009}$

The Matlab code for the conjugate gradient iterative method can be formed in Appendix H.

4.1 Conclusion

In this thesis we have used two methods to solve homogeneous and inhomogeneous parabolic partial differential equation subject to Dirichlet, Neumann and Robin's boundary conditions, these methods are finite difference method (FDM) and finite element method (FEM).

The discretization process converts the initial boundary value problem into n-algebraic linear equations. This system has been solved by several iterative schemes. These are: Jacobi, Gauss-Seidel, Successive over Relaxation and conjugate gradient method.

We observe that the finite difference method is very simple and efficient method for approximating the solution of initial boundary value problem when the domain has regular shape, while the finite element method is more efficient for irregular domain. Moreover, we clearly see that the conjugate gradient method is one of the most efficient and accurate method

in comparison with the other iterative techniques. It requires less number of iterations and least error.

References

[1] F. Abdelnour, H. Voss and A. Raj, **Network Diffusion Accurately Models the Relationship between Structural and Functional Brain Connectivity Networks**, NCBI, 2014.

[2] H. Abood, F. Fadhel and L. Hammza, **The Existence and Uniqueness Solution to the Diffusion Equation by using Arbitrary Function,** University of Babylon, 2014.

[3] M. Asadzadeh, **an Introduction to the Finite Element Method (FEM) for Differential Equations**, December 10, 2014.

[4] I. Babuska and T. Stroubulis, **the Finite Element Method and its Reliability**, Clarendon Press, Oxford, 2001.

[5] M. Benzi, **Key Moments in the History of Numerical Analysis**, Emory University, Atlanta, GA, 2005.

[6] R. Boucekkine, C. Camacho and G. Fabbri, ***Spatial Dynamics and Convergence: The Spatial AK Model, Elsevier,* Journal of Economic Theory**, 2013.

[7] R. L. Burden and J. D. Faires, **Numerical Analysis**, Ninth Edition, Books Publishing Company, 2011.

[8] S. Chapra, **Applied Numerical Method with MATLAB for Engineers**, Published by McGraw-Hill Companies, Third Edition, 2012.

[9] R. Clough, ***the Finite Element Method in Plane Stress Analysis*, American Journal of Civil Engineers**, 1960.

[10] J. Demmel, **Applied Numerical Linear Algebra**, SIAM, 1997.

[11] S. J. Farlow, **Partial Differential Equations for Scientists and Engineers**, John Willey & Sons, 1982.

[12] L. A. Hageman and D. M. Young, **Applied Iterative Methods**, Academic Press, 1981.

[13] C. Johnson, **Numerical Solution of Partial Differential Equations by the Finite Element Method**, Dover Publications, New York, 2009.

[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. Kersale, **Analytic Solution of Partial Differential Equations**, School of Mathematics, University of Leeds, 2003.

[16] M. Konstantinov, **Foundations of Numerical Analysis**, Second Edition, 2007.

[17] R. Kress, **Numerical Analysis**, Springer-Verlag, New York, 1998.

[18] M. A. Lau and S. P. Kuruganty, **Spread Sheets Implementations for Solving Boundary-Valve Problems in Electromagnetics**, Article 1, 2010.

[19] J. Mathews, **Matlab Programming Guidebook for Numerical Methods**, 2^{nd} Edition, 1992.

[20] S. Mazumder, **Numerical Methods for Partial Differential Equations**, the Ohio State University, 2015.

[21] G. Mazzola, G. Milmesiter and J. Weissmann, **Comprehensive Mathematics for Computer Scientists 2**, Springer, 2005.

[22] A. R. Michell and D. F. Griffith, **The Finite Difference Method in Partial Differential Equations**, John Willey & Sons, 1980.

[23] M. Necti, H. R. B. Orlande, M. J. colaco and R. M. Cotta, **Finite Difference Methods in Heat Transfer**, Second Edition, CRC Press, 2017.

[24] J. Nocedal and S. Wright, **Numerical Optimization**, Series in Operations Research, Springer Verlag, 1999.

[25] L. Olsen-Kettle, **Numerical Solution of Partial Differential Equations**, The University of Queensland, 2011.

[26] E. Ostertagova, O. Ostertag and J. Bocko, **Problems of Mechanics Described by Parabolic and Hyperbolic Differential Equations (of Second Order)**, MMAMS, Technical University of Kosice, 2011.

[27] N. Qatanani, ***Analysis of the Heat Equation with Non-Local Radiation Terms in a Non-Convex Diffuse and Grey Surfaces,*** **European Journal of Scientific Research**, 15(2), 245-254, 2006.

[28] C. Rycroft, **Iterative Methods for Linear Systems**, Lecture Notes, UC Berkeley, Mathematics Department, November 2007.

[29] Y. Saad, **Iterative Methods for Sparse Linear Systems**, Second Edition, Society for Industrial and Applied Mathematics (SIAM), 2003.

[30] G. D. Smith, ***Numerical Solutions of Partial Differential Equations***, **Clarendon Press**, Oxford, 1987.

[31] V. Thomee, **From Finite Difference to Finite Elements A Short History of Numerical Analysis of Partial Differential Equations**, Elsevier Science B. V., 2001.

[32] J. A. Trangenston, **Numerical Solution of Elliptic and Parabolic Partial Differential Equations**, Cambridge University Press, 2013.

[33] C. Vuik, **Iterative Solution Methods**, Mathematics and Computer Science, Delft Institue of Applied Mathematics, Morgan Kaufmann Publishers, San Francisco, 2001.

[34] G. A. Watson, **the History and Development of Numerical Analysis in Scotland: A Personal Perspective**, University of Dundee, 2009.

[35] Z. Wu, J. Yin and C. Wang, **Elliptic and Parabolic Equations**, Jilin University, China, 2006.

[36] T. Young and M. J. Mohlenkamp, **Introduction to Numerical Methods and MATLAB Programing for Engineers**, Ohio University, 2018.

[37] L. Zhilin, Z. Qiao and T. Tang, **Numerical Solution of Differential Equations Introduction to Finite Difference and Finite Element Methods**, Cambridge University Press, 2018.

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=[1.8 -0.4 0 0 0 0 0 0;-0.4 1.8 -0.4 0 0 0 0 0;0 -0.4 1.8 0 0 0 0 0;-1 0
0 1.8 -0.4 0 0 0 0;0 -1 0 -0.4 1.8 -0.4 0 0 0;0 0 -1 0 -0.4 1.8 0 0 0;0 0 0 -1 0
0 1.8 -0.4 0;0 0 0 0 -1 0 -0.4 1.8 -0.4;0 0 0 0 0 -1 0 -0.4 1.8];
b=[40;0;20;40;0;20;40;0;20];
%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=[1.8 -0.4 0 0 0 0 0 0 0;-0.4 1.8 -0.4 0 0 0 0 0 0;0 -0.4 1.8 0 0 0 0 0 0;-1 0
0 1.8 -0.4 0 0 0 0;0 -1 0 -0.4 1.8 -0.4 0 0 0;0 0 -1 0 -0.4 1.8 0 0 0;0 0 0 -1 0
0 1.8 -0.4 0;0 0 0 0 -1 0 -0.4 1.8 -0.4;0 0 0 0 0 -1 0 -0.4 1.8];
b=[40;0;20;40;0;20;40;0;20];
%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=[1.8 -0.4 0 0 0 0 0 0;-0.4 1.8 -0.4 0 0 0 0 0;0 -0.4 1.8 0 0 0 0 0;-1 0
0 1.8 -0.4 0 0 0 0;0 -1 0 -0.4 1.8 -0.4 0 0 0;0 0 -1 0 -0.4 1.8 0 0 0;0 0 0 -1 0
0 1.8 -0.4 0;0 0 0 0 -1 0 -0.4 1.8 -0.4;0 0 0 0 0 -1 0 -0.4 1.8];
b=[40;0;20;40;0;20;40;0;20];

%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=[1.8 -0.4 0 0 0 0 0 0 0;-0.4 1.8 -0.4 0 0 0 0 0 0;0 -0.4 1.8 0 0 0 0 0 0;-1 0
0 1.8 -0.4 0 0 0 0;0 -1 0 -0.4 1.8 -0.4 0 0 0;0 0 -1 0 -0.4 1.8 0 0 0;0 0 0 -1 0
0 1.8 -0.4 0;0 0 0 0 -1 0 -0.4 1.8 -0.4;0 0 0 0 0 -1 0 -0.4 1.8];
f=[40;0;20;40;0;20;40;0;20];

err=1.0;
format long
s=[0;0;0;0;0;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=[1.687 -0.45 0 0;-0.45 1.9 0 0;-1.112 0 1.788 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];
%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 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=[1.687 -0.45 0 0;-0.45 1.9 0 0;-1.112 0 1.788 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];
%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 G

```
% 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=[1.687 -0.45 0 0;-0.45 1.9 0 0;-1.112 0 1.788 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];

%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 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=[1.687 -0.45 0 0;-0.45 1.9 0 0;-1.112 0 1.788 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];

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 I

```
% 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=[1.4909 -0.45 0 0;-0.45 1.9 0 0;-1.08367 0 1.7159 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];
%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 J

```
% 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=[1.4909 -0.45 0 0;-0.45 1.9 0 0;-1.08367 0 1.7159 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];
%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 K

```
% 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=[1.4909 -0.45 0 0;-0.45 1.9 0 0;-1.08367 0 1.7159 -0.45;0 -1 -0.45 1.9];
b=[0.866;0.5;0;0];

%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 L

```
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=[1.4909 -0.45 0 0;-0.45 1.9 0 0;-1.08367 0 1.7159 -0.45;0 -1 -0.45 1.9];
f=[0.866;0.5;0;0];

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 M

```
% 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=[1.9 -0.45 0 0;-0.45 1.9 0 0;-1 0 1.9 -0.45;0 -1 -0.45 1.9];
b=[0.86683;0.51167;0.0016667;0.0033333];
%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 N

```
% 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=[1.9 -0.45 0 0;-0.45 1.9 0 0;-1 0 1.9 -0.45;0 -1 -0.45 1.9];
b=[0.86683;0.51167;0.0016667;0.0033333];
%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 O

```
% 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=[1.9 -0.45 0 0;-0.45 1.9 0 0;-1 0 1.9 -0.45;0 -1 -0.45 1.9];
b=[0.86683;0.51167;0.0016667;0.0033333];

%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 P

```
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=[1.9 -0.45 0 0;-0.45 1.9 0 0;-1 0 1.9 -0.45;0 -1 -0.45 1.9];
f=[0.86683;0.51167;0.0016667;0.0033333];

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
```

جامعة النجاح الوطنية

كلية الدراسات العليا

طرق عددية لحل المعادلة التفاضلية الجزئية الخطية المكافئة

أعداد

سمير محمود مصلح

اشراف

أ.د. ناجي قطناني

قدمت هذه الاطروحة استكمالاً لمتطلبات الحصول على درجة الماجستير في الرياضيات، بكلية الدراسات العليا، في جامعة النجاح الوطنية، نابلس- فلسطين.

2019

طرق عددية لحل المعادلة التفاضلية الجزئية الخطية المكافئة

أعداد

سمير محمود مصلح

اشراف

أ.د. ناجي قطناني

الملخص

كثيراً من الظواهر الفيزيائية والطبيعية تظهر على شكل نماذج رياضية وتحديدا تظهر كمعادلات تفاضلية جزئية تصف طبيعة هذه الظواهر. في هذه الرسالة استخدمنا المعادلة التفاضلية الجزئية الخطية المكافئة من الدرجة الثانية بحيث يتم التركيز على معادلة الحرارة كنموذج لوصف تلك الظواهر.

في الواقع، فان معظم هذه المسائل يصعب حلها بطرق تحليلية. بدلا من ذلك، يمكن ان تحل عدديا باستخدام الاساليب الحسابية.

في هذه الاطروحة، معادلة الحرارة المتجانسة مع انواع مختلفة من الشروط الحدية تم حلها عدديا باستخدام طريقة الفروق المحدودة وطريقة العناصر المحدودة لتقريب الحل لمعادلة التفاضلية الجزئية المكافئة. وبهذا يتم تحويل المعادلة الى شكل اخر للوصول الى نظام خطي من المعادلات يمكن حلها باستخدام طرق تكرارية، مثل:

Jacobi, Gauss-Seidel, Successive Over Relaxation, Conjugate Gradient Methods

وعمل مقارنة بينهم.

وجدنا في هذا البحث من خلال ما بينته النتائج العددية ان طريقة الفروق المحدودة هي أكثر كفاءة من طريقة العناصر المحدودة للحصول على حل تقريبي للمعادلة وبأقل خطأ ممكن في حال كون المجال ذو اشكال هندسية منتظمة، وان طريقة العناصر المحدودة أكثر دقة للمجالات المعقدة والغير منتظمة. ايضا، نلاحظ ان الطريقة التكرارية Conjugate Gradient تعطي النتائج الاكثر دقة من بين الطرق التكرارية الاخرى.