Numerical Methods for Solving Elliptic Boundary-Value Problems

By
Mithqal Ghalib Yousef Naji

Supervised by
Dr. Samir Matar

Submitted in Partial Fulfilment of the Requirements for the degree of Master in Computational Mathematics, Faculty of Graduate Studies, at An-Najah National University, Nablus, Palestine.

2005
Numerical Methods for Solving Elliptic Boundary-Value Problems

By
Mithqal Ghalib Yousef Naji

This Thesis was defended successfully on 02/08/2005 and approved by

Committee Members

1. Dr. Samir Matar (Supervisor)
2. Dr. Mohammad Najib Ass'ad (Internal Examiner)
3. Dr. Anwar Salch (External Examiner)

Signature

\[ \text{Signature} \]

\[ 2/8/2005 \]
"قل هل يستوي الذين يعلمون والذين لا يعلمون"

صدق الله العظيم
TO MY PARENTS, SISTERS, BROTHERS, MY WIFE, MY SONS AND FRIENDS FOR THEIR MORAL SUPPORT
I would like to express my great thanks to Dr. Samir Matar who was continuously directing me to the last and successful steps in my research progress.

My thanks are also due to Dr Mohammad N. Ass'ad from the Mathematics Department.

Thanks are also due to Dr. Anwar Saleh from Arab American University for his advice and support.

Thanks are due to Mr. Nasha't Abu Ghalia for his kind cooperation in the critical times.

Thanks are due to Mr. Assem Hamamdeh for his support.

Special thanks go to my mother, father and wife for their help and encouragement.

Finally, I would like to express my utmost appreciation to my family, and my friends for all kinds of support, keen interest and concern.
CONTENTS

Abstract

1. Introduction. 1
   1.1 Study Approach. 3
   1.2 Model Problem. 3

2. Finite Difference Discretization of Elliptic Equation. 7
   2.1 Introduction. 8
   2.2 Difference Equations for Boundary Nodes 12
   2.3 Error Analysis. 16
   2.4 Rotated Five-Points Formula. 20
   2.5 A Seven - Point Formula. 23
   2.6 A Nine - Point Formula and Truncation Error. 29
   2.7 The Laplace Equation in Polar Coordinate. 33

3. Iterative Methods for Elliptic Equations 34
   3.1 Introduction. 35
   3.2 Jacobi Method. 35
   3.3 Gauss-Seidel Method. 39
   3.4 Successive Over-Relaxation Method. 40
   3.5 Multigrid Method 44
   3.5.1 Multigrid Method for Poisson's Equation in 2-D. 45
   3.5.2 Simple V-cycle Algorithm. 46
   3.5.3 Restricting the Residual to Coarse Lattice. 48
   3.5.4 Prolongation of the Correction to the Finer Lattice. 49
4. Computational Results 50
4.1 Introduction. 51
4.2 Numerical Algorithm. 51
4.3 Poisson's Equation. 67
4.4 Laplace's Equation in Polar Coordinates. 81
4.5 Conclusion and Results 84
Appendix 85
References 142
Numerical Methods for solving Elliptic Boundary value problems
By
Mithqal Ghalib Yousef Naji
Supervised by
Dr. Samir Matar

Abstract

Elliptic Partial Differential Equations of second order have been studied using some numerical methods. This type of differential equations has specific applications in physical and engineering models. In most applications, first-order and second-order formulas are used for the derivatives. In this work higher order formulas such as: seven-points and nine-points formulas are used. Using these formulas will transform the partial differential equation into finite difference equations. To solve the resulting finite difference equations the following iterative methods have been used: Jacobi method, Gauss-Seidel method, Successive Over-Relaxation method (SOR) and Multigrid method.

In this thesis, we found that multigrid methods are the most efficient among all other methods. The execution time for multigrid methods is of order three while the other methods is of order five.
Chapter One

Introduction
Introduction

The majority of the problems of physics and engineering fall naturally into one of the following three physical categories: equilibrium problems, eigenvalue problems and propagation problems.

Partial Differential Equations (PDEs), which are considered at the heart of many mathematical models used in engineering and physics, has given rise to extensive computations. Often the problems that one would like to solve exceed the capacity of even the most powerful computers. On the other hand, the time required is too large to all inclusion of advanced mathematical models in the design process.

The solutions of PDEs are important in many fields of science and engineering notably in electromagnetism, astronomy, and fluid dynamics, because they describe the behavior of electric, gravitational and fluid potential. Most of the PDEs that arise in mathematical models of physical phenomena are difficult (if not impossible) to solve analytically, so we have used numerical methods to approximate the solution.

Linear Second-Order Partial Differential Equations

The general form of a linear second order PDE in two-dimensions is:

\[ a \frac{\partial^2 u}{\partial x^2} + b \frac{\partial^2 u}{\partial x \partial y} + c \frac{\partial^2 u}{\partial y^2} + d \frac{\partial u}{\partial x} + e \frac{\partial u}{\partial y} + fu + g = 0 \]  

(1.1)

where \( a, b, c, d, e, f \), and \( g \) are coefficients which may be constants, or functions of the independent variables \( x \) and \( y \).
This linear second order PDE in two independent variables can be classified as one of three standard or canonical form which we identify as hyperbolic, parabolic, or elliptic. The classifications of the PDEs are:

- Hyperbolic if \( b^2 - 4ac > 0 \)
- Parabolic if \( b^2 - 4ac = 0 \)
- Elliptic if \( b^2 - 4ac < 0 \)

In this research, we compare different numerical methods for solving linear elliptic boundary-value problems (bvps). We use several different methods to solve the elliptic PDEs such as:

- Successive Over-Relaxation (SOR), Jacobi method, Gauss-Seidel, and Multigrid method.

1.1 Study Approach

The aim of this work is to:

study, analyze and develop some numerical methods for solving elliptic partial differential equations with boundary conditions.

Also to investigate the most efficient method among the different methods used. We used MATLAB as a computational tool.

1.2 Model Problem

Elliptic PDEs arise usually from equilibrium or steady-state problems. A typical example of elliptic equation in two dimensions is the well-known Helmholtz equation:
\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \lambda u = f(x, y) \tag{1.2}
\]

Important special cases:

-when \(\lambda=0\), the above equation leads to Poisson’s equation which is:

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \tag{1.3}
\]

-when \(\lambda=0\) and \(f = 0\) the above equation is called Laplace’s equation which is:

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{1.4}
\]

If the solution \(u(x, y)\) is satisfying, Poisson's equation in a square region

\[
S = \{ (x, y) \mid a < x < b, a < y < b \}
\]

with boundary conditions \(u(x, y) = g(x, y)\) on \(\partial S\), where \(\partial S\) is the boundary of \(S\).

Poisson’s equation summarizes the flow motion of incompressible viscous fluid, and the inverse square law theories of electricity, magnetism and gravity matter at points where the charged density pole strength or mass density respectively is non-zero.

The solution to an electrostatic problem is straight forward for the case in which the charge distribution is everywhere specified for them; the potential and electric field are given directly as integrals over this charge distribution.
\[ u = \frac{1}{4\pi \varepsilon_0} \int \frac{dq^\prime}{|r-r^\prime|} \]  

(1.5)

where \( q^\prime \) is the charge inside Gauss surface.

\( \varepsilon_0 \) is the permittivity of the air.

\(|r-r^\prime|\) is the distance between the charge and the point.

\[ E = \frac{1}{4\pi \varepsilon_0} \int \frac{(r-r^\prime)dq^\prime}{|r-r^\prime|} \]  

(1.6)

First, we have the differential form of Gauss’s law:

\[ \text{Diverge } E = \frac{1}{\varepsilon_0} \rho \]  

(1.7)

Furthermore, in a purely electrostatic field, \( E \) may be expressed as the negative gradient of the potential:

\[ E = -\nabla u \]  

(1.8)

Combining equation (1.7) and (1.8) we obtain:

\[ \text{div } (\nabla u) = \frac{1}{\varepsilon_0} \rho \]

It is convenient to think of \( \text{div } \nabla \) as a single differential operator \( \nabla \nabla \) or \( \nabla^2 \). Where \( \nabla^2 \) is called Laplacian operator.

\[ \nabla^2 u = -\frac{1}{\varepsilon_0} \rho \]  

(1.9)

and this is a Poisson’s equation.
The operator $\nabla^2$ involves differentiation with respect to more than one independent variable which may be solved once we know the functional dependence of $\rho(x,y)$, and the appropriate boundary conditions.

Poisson’s equation can be expressed in three dimensions as:

\[
\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -\frac{\rho}{\varepsilon_0} \quad \text{in rectangular coordinates or}
\]

\[
\begin{align*}
\nabla^2 u &= \frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial u}{\partial r} \right) + \frac{1}{r^2 \sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial u}{\partial \theta} \right) + \frac{1}{r^2 \sin \theta} \frac{\partial^2 u}{\partial \phi^2} = -\frac{\rho(r, \theta)}{\varepsilon_0} \\
\text{(1.10)}
\end{align*}
\]

in spherical coordinates, or

\[
\begin{align*}
\nabla^2 u &= \frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2} = -\frac{\rho(r, \theta, z)}{\varepsilon_0} \\
\text{(1.11)}
\end{align*}
\]

in cylindrical coordinates.
Chapter Two

Finite Difference Discretization of Elliptic Equation
2.1 Introduction

Many engineering applications involve boundary value problems that require solving elliptic partial differential equations (PDEs). The discretization of such boundary-value problems leads to linear system $A \mathbf{u} = \mathbf{f}$, where $\mathbf{u}$ is the set of unknowns corresponding to the unknown variables in the PDE and $\mathbf{f}$ is the set of discrete values of the known function in the PDE.

Boundary conditions

For elliptic PDEs, there are given boundary conditions where in some property of $u$ is specified $\partial S$. With either Laplace’s or Poisson’s equations, we can define three types of boundary conditions.

Types of boundary conditions

The solution in an interior points of the region $S$ depends on all the data given on the boundary $\partial S$. The conditions on this boundary are of three types:

Fig. (2.1) Illustration of the region for Elliptic PDEs
It is often the case that an elliptic boundary-value problem is specified by boundary conditions that are of different parts of $\partial S$.

**Numerical Solution for Boundary-Value Problem**

The methods that have been used are based on finite difference methods for solving linear boundary value problems.

1. Define discrete mesh points within the domain $S$ of boundary-value problems.
2. Replace derivatives in the given PDE by some finite differences.
3. Solve the system of linear equations at all mesh points.

Diagramatically, the above steps are shown:
Approximate solution is determined at all mesh points simultaneously by solving single system of linear equations.

Consider the elliptic boundary problem Eqs (1.4) with a square domain:

\[ S = \{(x, y) : 0 \leq x \leq a, \ 0 \leq y \leq a\} \quad (2.1) \]

Define the mesh point as the points of intersection of the straight lines

\[ x_i = ih, \ i = 1, 2, \ldots, n \quad \text{and} \quad y_j = jk, \ j = 1, 2, \ldots, m. \]

\[ h = \frac{a}{n} \quad \text{and} \quad k = \frac{c}{m}. \]

Using Taylor's theorem, the second order central–difference approximations to first and second derivatives at the interior mesh point \((i,j)\) are:

\[ \left( \frac{\partial u}{\partial x} \right)_{i,j} = \frac{u(i+1, j) - u(i-1, j)}{2h} + O(h^2) \quad (2.2.a) \]

\[ \left( \frac{\partial u}{\partial y} \right)_{i,j} = \frac{u(i, j+1) - u(i, j-1)}{2k} + O(k^2) \quad (2.2.b) \]

\[ \left( \frac{\partial^2 u}{\partial x^2} \right)_{i,j} = \frac{u(i+1, j) - 2u(i, j) + u(i-1, j)}{h^2} + O(h^2) \quad (2.2.c) \]

\[ \left( \frac{\partial^2 u}{\partial y^2} \right)_{i,j} = \frac{u(i, j+1) - 2u(i, j) + u(i, j-1)}{k^2} + O(k^2) \quad (2.5.d) \]

(Lapidus, 1982)
Fig.(2.2): Five-point scheme for Laplace equation.

Taking \( x_i = i \Delta x \) and \( y_j = j \Delta y \) the five points finite difference approximation to the elliptic PDE at the interior mesh point \((x_i, y_j)\) Fig. (2.2) is

\[
B_{ij} u_{i,j-1} + D_{ij} u_{i-1,j} + E_{ij} u_{i,j} + F_{ij} u_{i+1,j} + H_{ij} u_{i,j+1} = g_{i,j}.
\]  
\[ \begin{array}{c}
\text{for } i=1,2,...,n \\
\text{for } j=1,2,...,m
\end{array}\]  
(2.3)

where \( B_{i,j}, D_{i,j}, E_{i,j}, F_{i,j}, H_{i,j} \) and \( g_{i,j} \) are known values. In the case of Laplace's equation \( B, D, E, F \) and \( H \) are 1, 1, \(-4\), 1 and 1 respectively.

**Linear Systems:**

Defining the vector \( \mathbf{u} \) to be

\[
\begin{bmatrix}
\mathbf{u}_{1,1} \cdots \mathbf{u}_{1,M-1,1} ; \mathbf{u}_{1,2} \cdots \mathbf{u}_{1,M-1,2} ; \cdots ; \mathbf{u}_{1,M-1} \cdots \mathbf{u}_{1,M-1,M-1}
\end{bmatrix}^T
\]  
(2.4)

Where \([\cdot]^T\) denotes the transpose and imposes an ordering on the \((M-1)^2\) unknown grid values. With this ordering, the totality of equations at the \((M-1)^2\) internal nodes of the unit square leads to the matrix equation

\[
\mathbf{A} \mathbf{u} = \mathbf{b}
\]  
(2.5)

Where \( \mathbf{A} \) is a matrix of order \((M-1)^2\) given by:
\[ A = \begin{bmatrix}
   B & -J & 0 \\
   -J & B & -J \\
   \cdot & \cdot & \cdot \\
   \cdot & \cdot & \cdot \\
   -J & B & -J \\
   0 & -J & B
\end{bmatrix} \]  \quad (2.6)

\( M \) : the number of interior points in each direction

with \( J \) the identity matrix of order \((M-1)\) and \( B \) the following matrix of order \((M-1)\) given by

\[
B = \begin{bmatrix}
   4 & -1 & \cdot & \cdot & \cdot \\
   -1 & 4 & -1 & \cdot & \cdot \\
   \cdot & \cdot & \cdot & \cdot & \cdot \\
   \cdot & \cdot & \cdot & \cdot & \cdot \\
   -1 & 4 & -1 \\
   0 & -1 & 4
\end{bmatrix} \]  \quad (2.7)

The components of the vector \( b \) depend on the boundary values of \( g(x,y) \) at the grid points on the perimeter of the square region and the equation .(Mitchell, 1980)

2.2 Difference Equations for Boundary Nodes

(a) Dirichlet conditions:

When \( u(x,y) = g(x,y) \) on the boundary \( \partial S \), the nodal values are \( u_{i,j} = g(x_i,y_j) \) on \( \partial S \), also when the node \((i,j)\) is adjacent to the boundary then this node is either adjacent to one boundary node Fig (2.3.a) or
adjacent to two boundary nodes (Fig. 2.3.b).

In the first case let the boundary node be \((i+1, j)\) then the difference equation at the point \((x_i, y_j)\) can be written as:

\[
B_{i,j} u_{i,j-1} + D_{i,j} u_{i-1,j} + E_{i,j} u_{i,j} + H_{i,j} u_{i,j+1} = g_{i,j} - F_{i,j} u_{i+1,j}
\]  
\[
(2.8)
\]

where \(u_{i+1,j}\) is given (known value) by the boundary condition.

In the second case when the node \((i, j)\) is near the south-east corner of the boundary of a square region. Let the two known boundary nodes be \((i + 1, j)\) and \((i, j - 1)\) as in Fig. (2.3.b), then the finite difference equation for this point is:

\[
D_{i,j} u_{i-1,j} + E_{i,j} u_{i,j} + H_{i,j} u_{i,j+1} = g_{i,j} - B_{i,j} u_{i,j-1} - F_{i,j} u_{i+1,j}
\]  
\[
(2.9)
\]

where \(u_{i,j-1}\) and \(u_{i+1,j}\) are given by the boundary conditions.
(b) Neuman condition:

When \( \frac{\partial u}{\partial n} = V(x, y) \) is on the boundary, then in the five-point difference scheme, three of the nodes lie on the boundary line, one inside the boundary at \((i-1, j)\) and the fifth at \((i+1, j)\) as in Fig. (2.4).

![Fig. (2.4)](image)

To find the value of \( u(i+1, j) \), we use either the following forward difference approximation namely:

\[
V(x, y) = \frac{u_{i+1, j} - u_{i, j}}{h}
\]  

(2.10)

which gives:

\[
u_{i+1, j} = u_{i, j} + hV_{i, j},
\]

or the central difference approximation (2.5.a)

Using Eq. (2.13) the finite difference equation (2.6) at \((i, j)\) can be written as:
\begin{equation}
B_{i,j} u_{i,j-1} + D_{i,j} u_{i-1,j} + (E_{i,j} + F_{i,j}) u_{i,j} + H_{i,j} u_{i,j+1} = g_{i,j} - F_{i,j} V_{i,j} h.
\end{equation}

(see figure (2.2)).

Taking the central difference approximation \( \frac{\partial u}{\partial n} \), the five-point difference equation at \((i,j)\), Fig (2.4), can be written as:

\begin{equation}
B_{i,j} u_{i,j-1} + (D_{i,j} + F_{i,j}) u_{i-1,j} + E_{i,j} u_{i,j} + H_{i,j} u_{i,j+1} = g_{i,j} - 2h F_{i,j} V_{i,j}
\end{equation}

The totality of equations at the \((M+1)^2\) grid points of the square leads to the matrix equation

\begin{equation}
A u = 2h G,
\end{equation}

Where \( A \) is a matrix of order \((M+1)^2\) given by:

\[
A = \begin{bmatrix}
B & -2I & 0 \\
-I & B & -I \\
& & . \\
& & . \\
& & . \\
& & . \\
0 & -2I & B
\end{bmatrix}
\end{equation}

(2.14)

With \( I \) the identity matrix of order \((M+1)\), and \( B \) matrix of order \((M+1)\) given by:

\[
B = \begin{bmatrix}
4 & -2 & 0 \\
-1 & 4 & -1 \\
& & . \\
& & . \\
& & . \\
& & . \\
0 & -2 & 4
\end{bmatrix}
\end{equation}

(2.15)
The vectors $u$ and $G$ of eq's (2.12) are respectively given by:

$$[u_{0,0} \ldots u_{M,M},0; u_{0,1} \ldots u_{M,1}; \ldots; u_{0,M} \ldots u_{M,M}]^T$$

and

$$[2V_{0,0},V_{1,0} \ldots V_{M-1,0},2V_{M,0}; V_{0,1},0 \ldots 0,V_{M,1}; \ldots; V_{0,m-1},0 \ldots 0,V_{M,m-1}; 2V_{0,M},V_{1,M} \ldots V_{M-1,M},2V_{M,M}]^T$$

(Smith, 1978; Mitchel, 1969).

(c) Robbin's conditions

The boundary conditions of the form (2.3) can be incorporated into the difference equations for the boundary nodal points by an extension of the methods outlined in the Dirichlet and Neuman problems (a) and (b).

(Smith, 1978; Mitchel, 1969).

2.3 Error Analysis

The following is an error analysis of five points difference approximation to Laplace's equation over the region $S$.

Consider the Dirichlet problem:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (x,y) \text{ on } S \quad (2.16.a)$$

$$u(x,y) = g(x,y), \quad (x,y) \text{ on } \partial S \quad (2.16.b)$$

where $S$ is defined by equation (2.4) and $g$ is a known function.

Define the set of all interior mesh points in $S$ by $S_h$. At the points of $S_h$ we replace $\nabla^2 u(x,y)$ by the five-point difference approximation of Laplace's equation, namely,
\[ \nabla_h^2 u_h(x,y) = \frac{u_{i,j+1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j-1}}{h^2} \] (2.17)

where \( u_h(x,y) \) is the computed solution. To bound the truncation error
\[ |\nabla_h^2 u_h(x,y) - \nabla^2 u(x,y)| \] for all \((x_i, y_j) \in S_h\),

We use the Taylor's series expansions for the points \( u_{i+1,j}, u_{i-1,j}, u_{i,j+1}, u_{i,j-1} \), and \( u_{i,j+1} \):

\[
\begin{align*}
    u_{i+1,j} &= u + h \frac{\partial u}{\partial x} + \frac{1}{2} h^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{6} h^3 \frac{\partial^3 u}{\partial x^3} + \frac{1}{24} h^4 \frac{\partial^4 u(x + \beta h, y)}{\partial x^4} \\
    u_{i-1,j} &= u - h \frac{\partial u}{\partial x} + \frac{1}{2} h^2 \frac{\partial^2 u}{\partial x^2} - \frac{1}{6} h^3 \frac{\partial^3 u}{\partial x^3} + \frac{1}{24} h^4 \frac{\partial^4 u(x - \beta h, y)}{\partial x^4}
\end{align*}
\] (2.18) (2.19)

Where \( 0 < \beta < 1 \).

Adding these two equations leads to
\[
\begin{align*}
    u_{i+1,j} - 2u_{i,j} + u_{i-1,j} &= (h^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{12} h^4 \frac{\partial^4 u(x + \beta h, y)}{\partial x^4})_{i,j}.
\end{align*}
\] (2.20)

Similarly, for \( u_{i,j+1} \) and \( u_{i,j-1} \) we get that
\[
\begin{align*}
    u_{i,j+1} - 2u_{i,j} + u_{i,j-1} &= (h^2 \frac{\partial^2 u}{\partial y^2} + \frac{1}{12} h^4 \frac{\partial^4 u(x, y + \gamma h)}{\partial y^4})_{i,j}
\end{align*}
\] (2.21)

where \( 0 < \gamma < 1 \).

Adding (2.23) and (2.24) we get:
\[
|\nabla_h^2 u_h(x,y) - \nabla^2 u(x,y)| = \frac{h^2}{12} \left[ \frac{\partial^4 u(x + \beta h, y)}{\partial x^4} + \frac{\partial^4 u(x, y + \gamma h)}{\partial y^4} \right].
\] (2.22)

Hence if \( u(x,y) \) has bounded derivatives of all order up to and including the fourth in \( \bar{S} = S \cup \partial S \), then for all \((x, y) \in S_h\),
\[
|\nabla_h^2 u_h(x,y) - \nabla^2 u(x,y)| \leq \frac{h^2}{6} M_4
\] (2.23)
Where \( u(x,y) \) is the solution of problem (2.19) and

\[
M_4 = \max \{ \max_{(x,y) \in S} \left| \frac{\partial^4 u(x,y)}{\partial x^4} \right|, \max_{(x,y) \in S} \left| \frac{\partial^4 u(x,y)}{\partial y^4} \right| \}. \tag{2.24}
\]

To find a bound for error in \( u_h(x,y) \) which is called discretization error \( e_h \), we apply a Theorem (Smith, 1978; p.223) which states that:

If \( V \) is any function defined on the set of mesh points \( s_h \) in the square region \( S \) (2.19) then

\[
\max_{s_h} |V| \leq \max_{s_h} |V| + \frac{1}{4}(a^2 + a^2) \max_{s_h} |\nabla^2 V - \nabla^2 |.
\]

Applying this theorem to the discretization error \( e_h \), we get:

\[
\max_{s_h} |e_h| \leq \max_{s_h} |e_h| + \frac{1}{4}(a^2 + a^2) \max_{s_h} |\nabla^2 V - \nabla^2 |.
\]

but \( e_h = 0 \) on \( \partial S_h \) because \( u_h + u = g \) on \( \partial S_h \) by Eq. (2.19.b). Hence, by Eq.(2.26), we conclude that

\[
\max_{s_h} |e_h| \leq \frac{1}{24}(a^2 + a^2) h^2 M_4.
\]

Therefore \( u_h \) converges to the exact solution \( u \) as \( h \) tends to zero and the discretization error is proportional to \( h^2 \) (Smith, 1978; Whiteman, 1982).

**Example:**

Consider Laplace's equation

\[
u_{xx} + u_{yy} = 0
\]
On unit square with boundary conditions \( u(x,y) = \begin{cases} 
0 : x = 0 \\
0 : x = 1 \\
0 : y = 0 \\
1 : y = 1 
\end{cases} \) as shown below

![Diagram](image)

Define a discrete mesh in this domain, including boundaries with \( n = 2 \), \( h = \frac{1}{3} \), and \( k = \frac{1}{3} \), as shown on right above

Interior grid points where we will compute approximate solution are given by:

\[(x_i, y_j) = (ih, ky), \quad i, j = 1, 2, ..., n\]

After replacing the derivatives by central difference approximation at each interior mesh point we get the finite difference equation

\[
\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0
\]

where \( u_{i,j} \) approximates the true solution \( u(x_i, y_j) \) for \( i, j = 1, 2 \)

The point \((x_i, y_j)\) is a boundary point if \( i, j = 0 \) or \( i, j = 3 \)

The result is the following linear system:
The above equations can be written in matrix form as:

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

where

$$ A = \begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} , \quad \mathbf{u} = \begin{bmatrix} u_{i,1} \\ u_{i,2} \\ u_{i,3} \\ u_{i,4} \end{bmatrix} , \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} $$

Using Gaussian elimination with backward substitution or any direct method, gives the exact solution:

$$ \mathbf{u} = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.375 \\ 0.375 \end{bmatrix} $$

2.4 Rotated Five-Points Formula

Another type of discretization, so-called rotated discretization, can be achieved by rotating the $i$- and $j$- plane axis clockwise by $45^\circ$. Thus, the discretized form of Laplace’s equation at the mesh point $(x_i, y_j)$ with the finite differences formula will lead to the rotated five points scheme as:

![Rotated Five-Points Scheme](image-url)
Now Laplace's equation can be written as:

\[ u_{xx} + u_{yy} = \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 + \alpha_4 u_4 - \alpha_0 u_0 = 0 \]

After we solve the above equations for \( \alpha \)'s we obtain

\[ \alpha_0 = \frac{2}{k^2}, \quad \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{2k^2}. \]

For the above case, we need only to define a new \( k' \) as:

\[ k' = \frac{k}{(2)^{1/2}} \]

so Laplace's equation can be written as:

\[ u_{xx} + u_{yy} = (u_1 + u_2 + u_3 + u_4 - 4u_0) = 0 \]
Now let's write the above equation in a vector matrix form:

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

Let's take $(n = m = 4)$ in each direction

For node $(1, 1)$:

$$ u_{1,1} = \frac{1}{4}[u_{2,2} + u_{0,2} + u_{2,0} + u_{0,0}] $$

For node $(2, 1)$:

$$ u_{2,1} = \frac{1}{4}[u_{3,2} + u_{1,2} + u_{3,0} + u_{1,0}] $$

For node $(3, 1)$:

$$ u_{3,1} = \frac{1}{4}[u_{4,2} + u_{2,2} + u_{4,0} + u_{2,0}] $$

For node $(1, 2)$:

$$ u_{1,2} = \frac{1}{4}[u_{2,3} + u_{0,3} + u_{2,1} + u_{0,1}] $$

For node $(2, 2)$:

$$ u_{2,2} = \frac{1}{4}[u_{3,3} + u_{1,3} + u_{3,1} + u_{1,1}] $$

For node $(3, 2)$:

$$ u_{3,2} = \frac{1}{4}[u_{4,3} + u_{2,3} + u_{4,1} + u_{2,1}] $$

For node $(1, 3)$:

$$ u_{1,3} = \frac{1}{4}[u_{2,4} + u_{0,4} + u_{2,2} + u_{0,2}] $$

For node $(2, 3)$:

$$ u_{2,3} = \frac{1}{4}[u_{3,4} + u_{1,4} + u_{3,2} + u_{1,2}] $$
For node (3,3): 
\[ u_{3,3} = \frac{1}{4}[u_{4,4} + u_{2,4} + u_{2,2} + u_{4,2}] \]

\[
A = \begin{bmatrix}
4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 4 & 0 & 0 & 0 & -1 & 0 \\
-1 & 0 & -1 & 0 & 4 & 0 & -1 & 0 & -1 \\
0 & -1 & 0 & 0 & 0 & 4 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\
\end{bmatrix},
\]

\[
u = \begin{bmatrix} u_{i,j} \end{bmatrix}^T
\]

\[
b = [400 \ 600 \ 700 \ 100 \ 0 \ 0 \ 200 \ 200 \ 500]^T
\]

the above system can be solved and the approximated solution will be:

\[
u = [137.5 \ 187.9808 \ 2125 \ 88.4615 \ 15000 \ 63.4615 \ 87.500 \ 65.8654 \ 1625]^T
\]

2.5 A Seven-Point Formula

We begin this discussion by developing a series of seven point finite difference approximations for the model elliptic PDE.

We consider the interior of S but ignore in the present, the boundary conditions. The subscripts on \( u \) will be \( i \) and \( j \) respectively.

In a more general sense, we may desire a seven–point finite formula that either uses \( u_0 \) and any sixth other points or has variable spacing throughout or in some specific region of \( S \). This can be done using Laplace’s equation as an illustration, by setting

\[ u_{xx} + u_{yy} = \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 + \alpha_4 u_4 + \alpha_5 u_5 + \alpha_6 u_6 - \alpha_0 u_0 \]
Where the $u_1,\ldots, u_6$ are the values of $U$ selected mesh points around $U_0$ and $\alpha_1,\ldots, \alpha_6$ are parameters to be determined by Taylor's

\[ u_1 = \left[ u_0 + (s_1 k)u_{x_1} + \frac{(s_2 k)^2}{2} u_{x_2} + \frac{(s_1 k)^3}{6} u_{x_3} + \frac{(s_1 k)^4}{24} u_{x_4} + \ldots \right]_0 \]

\[ u_3 = \left[ u_0 - (s_3 k)u_{y_1} + \frac{(s_3 k)^2}{2} u_{y_2} - \frac{(s_3 k)^3}{6} u_{y_3} + \frac{(s_3 k)^4}{24} u_{y_4} - \ldots \right]_0 \]

\[ u_2 = \left[ u_0 - (s_2 k)u_{x_1} + \frac{(s_2 k)^2}{2} u_{x_2} - \frac{(s_2 k)^3}{6} u_{x_3} + \frac{(s_2 k)^4}{24} u_{x_4} + \ldots \right]_0 \]

\[ u_4 = \left[ u_0 + (s_4 k)u_{y_1} + \frac{(s_4 k)^2}{2} u_{y_2} + \frac{(s_4 k)^3}{6} u_{y_3} + \frac{(s_4 k)^4}{24} u_{y_4} + \ldots \right] \]

\[ u_5 = \left[ u_0 + (s_5 k)u_{x_1} + (s_5 k)u_{y_1} + \frac{(s_5 k)^2}{2} u_{x_2} + \frac{(s_5 k)^2}{2} u_{y_2} + \frac{(s_5 k)^3}{6} u_{x_3} + \frac{(s_5 k)^3}{6} u_{y_3} + \ldots \right] \]
\[ u_6 = [u_0 - (s_6 k) u_{x^1} - (s_6 k) u_{y^1} + \frac{(s_6 k)^2}{2} u_{y^2} + \frac{(s_6 k)^2}{2} u_{x^3} + \\
(s_6 k)^2 u_{x^1 y^1} - \frac{(s_6 k)^3}{6} u_{x^1} - \frac{(s_6 k)^3}{2} u_{x^3 y^1} - \frac{(s_6 k)^3}{6} u_{x^1 y^3} - \frac{(s_6 k)^3}{6} u_{y^3} + ... ] \\
u_{x^1} + u_{y^3} = u_0[-\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6] + \\
u_{x^1} [\alpha_1(s_1 k) - \alpha_2(s_2 k) + \alpha_5(s_5 k) - \alpha_6(s_6 k)] + \\
u_{y^3} [\alpha_1(s_1 k) + \alpha_4(s_4 k) + \alpha_5(s_5 k) - \alpha_6(s_6 k)] + \\
\frac{u_{x^1}^2}{2} [\alpha_1(s_1 k)^2 + \alpha_2(s_2 k)^2 + \alpha_5(s_5 k)^2 + \alpha_6(s_6 k)^2] + \\
\frac{u_{y^3}^2}{2} [\alpha_3(s_3 k)^2 + \alpha_4(s_4 k)^2 + \alpha_5(s_5 k)^2 + \alpha_6(s_6 k)^2] + \\
\frac{u_{x^1}^3}{6} [\alpha_2(s_1 k)^3 - \alpha_2(s_2 k)^3 + \alpha_3(s_3 k)^3 + \alpha_5(s_5 k)^3 - \alpha_6(s_6 k)^3] + \\
\frac{u_{y^3}^3}{6} [-\alpha_3(s_3 k)^3 + \alpha_4(s_4 k)^3 + \alpha_5(s_5 k)^3 - \alpha_6(s_6 k)^3] + \\
\frac{u_{x^1}^4}{24} [\alpha_1(s_1 k)^4 + \alpha_2(s_2 k)^4 + \alpha_5(s_5 k)^4 + \alpha_6(s_6 k)^4] + \\
\frac{u_{y^3}^4}{24} [\alpha_3(s_3 k)^4 + \alpha_4(s_4 k)^4 + \alpha_5(s_5 k)^4 + \alpha_6(s_6 k)^4] \\
\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 - \alpha_0 = 0 \\
\alpha_1(s_1 k) - \alpha_2(s_2 k) + \alpha_5(s_5 k) - \alpha_6(s_6 k) = 0 \\
- \alpha_3(s_3 k) + \alpha_4(s_4 k) + \alpha_5(s_5 k) - \alpha_6(s_6 k) = 0 \\
\alpha_1(s_1 k)^2 + \alpha_2(s_2 k)^2 + \alpha_5(s_5 k)^2 + \alpha_6(s_6 k)^2 = 2 \\
\alpha_3(s_3 k)^2 + \alpha_4(s_4 k)^2 + \alpha_5(s_5 k)^2 + \alpha_6(s_6 k)^2 = 2 \\
\alpha_1(s_1 k)^3 - \alpha_2(s_2 k)^3 + \alpha_5(s_5 k)^3 - \alpha_6(s_6 k)^3 = 0 \]
\[-\alpha_3 (s_3 k)^3 + \alpha_4 (s_4 k)^3 + \alpha_5 (s_5 k)^3 - \alpha_6 (s_6 k)^3 = 0\]

<table>
<thead>
<tr>
<th>(k)</th>
<th>(a_0)</th>
<th>(a_1)</th>
<th>(a_2)</th>
<th>(a_3)</th>
<th>(a_4)</th>
<th>(a_5)</th>
<th>(a_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(s_3)</td>
<td>(-s_2)</td>
<td>0</td>
<td>0</td>
<td>(s_3)</td>
<td>(-s_6)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(-s_3)</td>
<td>(s_4)</td>
<td>(s_3)</td>
<td>(-s_6)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(s_3 k^2)</td>
<td>(s_2 k^2)</td>
<td>0</td>
<td>0</td>
<td>(s_3 k^2)</td>
<td>(s_6 k^2)</td>
<td>(s_3 k^2)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(s_3 k^2)</td>
<td>(s_4 k^2)</td>
<td>(s_3 k^2)</td>
<td>(s_6 k^2)</td>
<td>(s_3 k^2)</td>
</tr>
<tr>
<td>0</td>
<td>(-s_3 k^3)</td>
<td>(-s_2 k^3)</td>
<td>0</td>
<td>0</td>
<td>(-s_3 k^3)</td>
<td>(-s_6 k^3)</td>
<td>(-s_3 k^3)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(-s_3 k^3)</td>
<td>(s_4 k^3)</td>
<td>(-s_3 k^3)</td>
<td>(-s_6 k^3)</td>
<td>(-s_3 k^3)</td>
</tr>
</tbody>
</table>

For regular region:

\(s_1 = s_2 = s_3 = s_4 = s_5 = s_6 = 1\)

Let \(\alpha_6 = r = 1\)

\(\alpha_5 = t = 1\)

\(\alpha_0 + \alpha_5 + \alpha_6 = 400\)

\(\alpha_1 + \alpha_5 = 100\)

\(\alpha_2 + \alpha_6 = 100\)

\(\alpha_3 + \alpha_6 = 100\)

\(\alpha_4 + \alpha_5 = 100\)

When we solve the above equations, we obtain:

\(\alpha_0 = 398, \alpha_1 = 99, \alpha_2 = 99, \alpha_3 = 99, \alpha_4 = 99, \alpha_6 = \alpha_5 = 1\)

\[u_{xx} + u_{yy} = 99u_1 + 99u_2 + 99u_3 + 99u_4 + u_5 + u_6 - 398u_0 = 0\]

\[u_0 = \frac{1}{398} (99u_1 + 99u_2 + 99u_3 + 99u_4 + u_5 + u_6)\]
Let us write the above equation in a matrix form as:

\[ A \mathbf{u} = \mathbf{b} \]

For node (1, 1):
\[ u_{1,1} = \frac{1}{398} \left[ 99(u_{2,1} + u_0 + u_{1,2}) + u_{2,2} + u_0 \right] \]

For node (2, 1):
\[ u_{2,1} = \frac{1}{398} \left[ 99(u_{3,1} + u_{1,1} + u_0 + u_{2,2}) + u_{3,2} + u_0 \right] \]

For node (3, 1):
\[ u_{3,1} = \frac{1}{398} \left[ 99(u_{4,1} + u_{2,1} + u_0 + u_{3,2}) + u_{4,2} + u_0 \right] \]

For node (4, 1):
\[ u_{4,1} = \frac{1}{398} \left[ 99(u_0 + u_{3,1} + u_0 + u_{4,2}) + u_0 + u_0 \right] \]

For node (1, 2):
\[ u_{1,2} = \frac{1}{398} \left[ 99(u_{2,2} + u_{0,2} + u_{1,1} + u_{1,3}) + u_{2,3} + u_{0,1} \right] \]

For node (2, 2):
\[ u_{2,2} = \frac{1}{398} \left[ 99(u_{3,2} + u_{1,2} + u_{2,1} + u_{2,3}) + u_{3,3} + u_{1,1} \right] \]

For node (3, 2):
\[ u_{3,2} = \frac{1}{398} \left[ 99(u_{4,2} + u_{2,2} + u_{3,1} + u_{3,3}) + u_{4,3} + u_{2,1} \right] \]
For node (4, 2): \[ u_{4,2} = \frac{1}{398} [99(u_0 + u_{3,2} + u_{4,1} + u_{4,3}) + u_0 + u_{3,1}] \]

For node (1, 3): \[ u_{1,3} = \frac{1}{398} [99(u_{2,3} + u_0 + u_{1,2} + u_{1,4}) + u_{2,4} + u_0] \]

For node (2, 3): \[ u_{2,3} = \frac{1}{398} [99(u_{3,3} + u_{1,3} + u_{2,2} + u_{2,4}) + u_{3,4} + u_{1,2}] \]

For node (3, 3): \[ u_{3,3} = \frac{1}{398} [99(u_{4,3} + u_{2,3} + u_{3,2} + u_{3,4}) + u_{4,4} + u_{2,3}] \]

For node (4, 3): \[ u_{4,3} = \frac{1}{398} [99(u_{3,3} + u_0 + u_{4,2} + u_{4,4}) + u_0 + u_{3,2}] \]

For node (1, 4): \[ u_{1,4} = \frac{1}{398} [99(u_{2,4} + u_0 + u_{1,3} + u_0) + u_0 + u_0] \]

For node (2, 4): \[ u_{2,4} = \frac{1}{398} [99(u_{3,4} + u_{2,3} + u_{4,4} + u_0) + u_0 + u_{1,3}] \]

For node (3, 4): \[ u_{3,4} = \frac{1}{398} [99(u_{4,4} + u_{2,4} + u_{3,3} + u_0) + u_0 + u_{2,3}] \]

For node (4, 4): \[ u_{4,4} = \frac{1}{398} [99(u_0 + u_{3,4} + u_{4,3} + u_0) + u_0 + u_{3,3}] \]

\[ A = \begin{bmatrix} 398 & -99 & 0 & 0 & -99 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -99 & 398 & -99 & 0 & 0 & -99 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -99 & 398 & -99 & 0 & 0 & -99 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -99 & 398 & 0 & 0 & -99 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -99 & 0 & 0 & 0 & 398 & -99 & 0 & 0 & -99 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -99 & 0 & 0 & -99 & 398 & -99 & 0 & 0 & -99 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -99 & 0 & 0 & -99 & 398 & -99 & 0 & 0 & -99 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -99 & 0 & 0 & -99 & 398 & 0 & 0 & -99 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -99 & 0 & 0 & 0 & 398 & -99 & 0 & 0 & -99 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -99 & 0 & 0 & -99 & 398 & -99 & 0 & 0 & -99 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -99 & 0 & 0 & -99 & 398 & 0 & 0 & 0 & -99 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -99 & 0 & 0 & -99 & 398 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -99 & 0 & 0 & -99 & 398 & -99 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -99 & 0 & 0 & -99 & 398 & -99 \end{bmatrix} \]
The above matrix can be written as:

\[
\begin{bmatrix}
A_1 & B_1 & 0 \\
D & A_2 & B_2 \\
0 & B_2^T & A_3
\end{bmatrix}
\]

This is Tridiagonal Block Matrix.

### 2.6 A Nine-points formula and Truncations Error

In this section, we develop higher-order nine-point approximations.

One approach is to use higher order approximation for \(U_{xx}\) and \(U_{yy}\).

We turn to the set of points shown in the figure above to drive this approximation. We may follow our previous procedure and estimate \(\alpha_0, \alpha_1, ..., \alpha_4\) and \(\alpha_5, ..., \alpha_8\) in the form

\[u_{xx} + u_{yy} = -\alpha_0 u_0 + \sum_{i=1}^{4} \alpha_i u_i + \sum_{i=5}^{8} \alpha_i u_i\]

For Laplace's equation. Substituting Taylor's series
\[ u_1 = [u_0 + (sk_1)u_{x_1} + \frac{(sk_1)^2}{2}u_{x_2} + \frac{(sk_1)^3}{6}u_{x_3} + \frac{(sk_1)^4}{24}u_{x_4} + \ldots]_0 \]

\[ u_2 = [u_0 + (sk_1)u_x + \frac{(sk_1)^2}{2}u_{x_2} + \frac{(sk_1)^3}{6}u_{x_3} + \frac{(sk_1)^4}{24}u_{x_4} + \ldots] \]

\[ u_3 = [u_0 - (s k_1)u_x + \frac{(sk_1)^2}{2}u_{x_2} - \frac{(sk_1)^3}{6}u_{x_3} + \frac{(sk_1)^4}{24}u_{x_4} + \ldots] \]

\[ u_4 = [u_0 - (sk_1)u_x + \frac{(sk_1)^2}{2}u_{y_2} - \frac{(sk_1)^3}{6}u_{x_3} + \frac{(sk_1)^4}{24}u_{x_4} + \ldots] \]

\[ u_5 = u_0 + (sk_2)u_x + (sk_2)u_y + \frac{(sk_2)^2}{2}u_{x_2} + \frac{(sk_2)^3}{2}u_{y_2} + (sk_2)^2u_{x_3} + \frac{(sk_2)^3}{6}u_{x_4} + \frac{(sk_2)^3}{6}u_{y_3} + \frac{(sk_2)^3}{2}u_{x_3} + \frac{(sk_2)^3}{2}u_{y_3} + \ldots \]

\[ u_6 = u_0 - (sk_2)u_x + (sk_2)u_y + \frac{(sk_2)^2}{2}u_{x_2} - \frac{(sk_2)^3}{2}u_{x_3} + \frac{(sk_2)^3}{2}u_{y_3} + \frac{(sk_2)^3}{2}u_{x_4} + \frac{(sk_2)^3}{2}u_{y_4} + \ldots \]

\[ u_7 = u_0 + (sk_2)u_x - (sk_2)u_y + \frac{(sk_2)^2}{2}u_{x_2} + \frac{(sk_2)^3}{2}u_{y_2} - (sk_2)^2u_{x_3} + \frac{(sk_2)^3}{6}u_{x_4} - \frac{(sk_2)^3}{6}u_{y_3} + \frac{(sk_2)^3}{2}u_{x_4} + \frac{(sk_2)^3}{2}u_{y_4} + \ldots \]

\[ u_8 = u_0 - (sk_2)u_x - (sk_2)u_y + \frac{(sk_2)^2}{2}u_{x_2} + \frac{(sk_2)^3}{2}u_{y_2} + (sk_2)^2u_{x_3} - \frac{(sk_2)^3}{6}u_{x_4} - \frac{(sk_2)^3}{6}u_{y_3} + \frac{(sk_2)^3}{2}u_{x_4} + \frac{(sk_2)^3}{2}u_{y_4} + \ldots \]

Collecting terms and solving for the \( \alpha_i \). We obtain for \( k_1 = k_2 = k \),

\[ \alpha_0 = \frac{10}{3k^2}, \quad \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{2}{3k^2}, \quad \alpha_5 = \alpha_6 = \alpha_7 = \alpha_8 = \frac{1}{6k^2}. \]

(Lapidus, 1982; Pinder, 1982)
Thus a forth-order approximation for Laplace's equation is given by:

\[ u_0 = \frac{1}{20} \left[ 4(u_1 + u_2 + u_3 + u_4) + (u_5 + u_6 - u_1 + u_8) \right] \]

Let's write the above equation in a matrix form,

First, assume that \( n = 5 \) in each direction and the function in each side equals zero.

For node (1): 
\[ u_1 = \frac{1}{20} \left[ 4(u_2 + u_8 + 0 + 0) + (u_7 + 0 - 0 + 0) \right] \]

For node (2): 
\[ u_2 = \frac{1}{20} \left[ 4(u_3 + u_7 + u_4 + 0) + (u_6 + u_8 - 0 + 0) \right] \]

For node (3): 
\[ u_3 = \frac{1}{20} \left[ 4(u_4 + u_6 + u_2 + 0) + (u_5 + u_7 - 0 + 0) \right] \]

For node (4): 
\[ u_4 = \frac{1}{20} \left[ 4(u_5 + u_3 + 0 + 0) + (u_6 + 0 - 0 + 0) \right] \]

For node (5): 
\[ u_5 = \frac{1}{20} \left[ 4(u_{12} + u_6 + u_4 + 0) + (u_{11} + 0 - u_2 + 0) \right] \]
The above linear equations can be written in a matrix form as:

\[ A \mathbf{u} = \mathbf{b}, \]

\[ \mathbf{u} = [u_1 \ u_2 \ u_3 \ u_4 \ u_5 \ u_6 \ u_7 \ u_8 \ u_9 \ u_{10} \ u_{11} \ u_{12} \ u_{13} \ u_{14} \ u_{15} \ u_{16}]^T \]

\[ \mathbf{b} = [1700 \ 1200 \ 1200 \ 2200 \ 1200 \ 0 \ 0 \ 200 \ 200 \ 0 \ 0 \ 1200 \ 1700 \ 600 \ 600 \ 600]^T \]

the above system can be solved and the approximated solution will be:

\[ \mathbf{u} = [1348 \ 14561 \ 15711 \ 17471 \ 13949 \ 10778 \ 9015 \ 7835 \ 623 \ 7428 \ 9364 \ 13083 \ 1292 \ 922 \ 782 \ 668]^T \]
### 2.7 The Laplace equation in polar coordinate

The Laplace equation in polar coordinate is written as:

\[
\nabla^2 u = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} = 0
\]  

(2.25)

Index notation in polar coordinate.

Let \( i \) and \( j \) be the index in the \( r \) and \( \theta \) directions respectively, the finite difference form of the Laplace equation is written as:

\[
\nabla^2 u = \frac{u(i-1,j) - 2u(i,j) + u(i+1,j)}{\Delta r^2} + \frac{1}{r_i} \frac{u(i+1,j) - u(i-1,j)}{2\Delta r} + \\
\frac{1}{r_i^2} \frac{u(i, j-1) - 2u(i, j) + u(i, j+1)}{\Delta \theta^2} = 0
\]

The value at the node \((i,j)\) can be computed from

\[
2 \left[ \frac{1}{\Delta r^2} + \frac{1}{r_i^2 \Delta \theta} \right] u(i,j) = \frac{u(i-1,j) + u(i+1,j)}{\Delta r^2} + \frac{1}{r_i} \frac{u(i+1,j) - u(i-1,j)}{2\Delta r} + \\
\frac{1}{r_i^2} \frac{u(i, j-1) + u(i, j+1)}{\Delta \theta^2}.
\]
Chapter Three

Iterative Methods for Solving Elliptic Equations
3.1 Introduction

In chapter two we discussed alternative methods for approximating the solution of linear elliptic equations, the finite difference method, using the differential equation. Each of these methods gives rise to a system of linear algebraic equations, which may be very large depending on the accuracy needed. A two-dimensional problem may lead to a system of several thousand unknowns, are common in real engineering situations. The solution of each a system is a major problem in itself and has been the subject of much detailed study. The most obvious property of the system is that it is extremely sparse. Even when there are many thousand unknowns each equation will involve one unknown and the unknowns at its immediate neighbours. In this chapter, we deals with some iterative methods.

3.2 Jacobi Iterative method:

Consider a linear system: \( Au = F \)

Take the matrix \( A \) and consider a split into: \( A = D - L - U \), where \( D \) is diagonal and \( -L \) and \( -U \) are the strictly lower and upper triangular parts of \( A \).

The linear system \( Au = F \) can be written as:

\[
(D - L - U)x = F \\
Dx = (L + U)x + F \\
x = D^{-1}(L + U)x + D^{-1}F
\]

Let \( R_j = D^{-1}(L + U) \), \( R_j = D^{-1}(D - A) = I - D^{-1}A \)

\( R_j \) is called the error propagation or iteration matrix for Jacobi method.
Then, the iteration is:

\[ u^{(new)} = R_j u^{(old)} + D^{-1} F \]

or

\[ u^{(r+1)}_{i,j} = \frac{(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})^{(ik)} - h^2 f_{i,j}}{4} \]  \hspace{1cm} (3.1)

**Error propagation matrix:**

From the derivation:

\[ x = D^{-1}(L + U)x + D^{-1}F \]
\[ x = R_j + D^{-1}F \]  \hspace{1cm} (3.2)

The iteration is

\[ u^{(new)} = Ru^{(old)} + D^{-1}F \]  \hspace{1cm} (3.3)

Subtracting eq's (3.3),(3.4)

\[ x - u^{(new)} = R_j x + D^{-1}F - (R_j u^{(old)} + D^{-1}F) \]

Hence:

\[ x - u^{(new)} = R_j x - R_j u^{(old)} \]
\[ x - u^{(new)} = R_j (x - u^{(old)}) \]

Error propagation:

\[ e^{(new)} = R_j e^{(old)}, \quad R_j = I - D^{-1} A \]
Example: A square grid is shown below for the solution of the two dimensional Laplace’s equation based on the given boundary conditions

\[ \begin{align*} U_a &= 10 \\ U_b &= 40 \\ U_c &= 0 \\ U_d &= 20 \end{align*} \]

The nodes in a two-dimensional, steady-state conduction.

\[ u^{(k)}_{ij} = \text{the value of the } u_{ij} \text{at the } k^{th} \text{ iteration} \]

\[ u^{(k)}_{11} = \frac{1}{4} [u_d + u_{21}^{(k-1)} + u_{12}^{(k-1)} + u_d] \]

\[ u^{(k)}_{21} = \frac{1}{4} [u_d + u_{21}^{(k-1)} + u_{12}^{(k-1)} + u_d] \]

\[ u^{(k)}_{31} = \frac{1}{4} [u_{(k-1)}^{21} + u_{32}^{(k-1)} + u_c + u_d] \]

\[ u^{(k)}_{12} = \frac{1}{4} [u_d + u_b + u_{(k-1)}^{12} + u_{(k-1)}^{11}] \]

\[ u^{(k)}_{22} = \frac{1}{4} [u_{(k-1)}^{12} + u_b + u_{(k-1)}^{32} + u_{(k-1)}^{21}] \]

\[ u^{(k)}_{32} = \frac{1}{4} [u_{(k-1)}^{22} + u_b + u_c + u_{(k-1)}^{31}] \]

initial guess \( (k = 0) \) Let \( u^{(0)}_{ij} = 0 \) for all nodes
1\textsuperscript{st} Iteration (k = 1)

\[ u^{(1)}_{11} = \frac{1}{4} [u_a + u^{(0)}_{12} + u^{(0)}_{21} + u_d] = \frac{1}{4} [10 + 0 + 0 + 20] = 7.5 \]

\[ u^{(1)}_{21} = \frac{1}{4} [u^{(0)}_{11} + u^{(0)}_{22} + u^{(0)}_{31} + u_d] = \frac{1}{4} [0 + 0 + 0 + 20] = 5 \]

\[ u^{(1)}_{31} = \frac{1}{4} [u^{(0)}_{21} + u^{(0)}_{32} + u_c + u_d] = \frac{1}{4} [0 + 0 + 0 + 20] = 5 \]

\[ u^{(1)}_{12} = \frac{1}{4} [u_a + u_b + u^{(0)}_{22} + u^{(0)}_{31}] = \frac{1}{4} [10 + 40 + 0 + 20] = 12.5 \]

\[ u^{(1)}_{22} = \frac{1}{4} [u^{(0)}_{12} + u_b + u^{(0)}_{32} + u^{(0)}_{11}] = \frac{1}{4} [0 + 40 + 0 + 20] = 10 \]

\[ u^{(1)}_{32} = \frac{1}{4} [u^{(0)}_{22} + u_b + u_c + u^{(0)}_{31}] = \frac{1}{4} [0 + 40 + 0 + 20] = 10 \]

2\textsuperscript{nd} Iteration (k = 2)

\[ u^{(2)}_{11} = \frac{1}{4} [u_a + u^{(1)}_{12} + u^{(2)}_{21} + u_d] = \frac{1}{4} [10 + 12.5 + 5 + 20] = 11.875 \]

\[ u^{(2)}_{21} = \frac{1}{4} [u^{(1)}_{11} + u^{(1)}_{22} + u^{(1)}_{31} + u_d] = \frac{1}{4} [7.5 + 10 + 5 + 20] = 10.625 \]

\[ u^{(2)}_{31} = \frac{1}{4} [u^{(1)}_{21} + u^{(1)}_{32} + u_c + u_d] = \frac{1}{4} [5 + 10 + 0 + 20] = 8.75 \]

\[ u^{(2)}_{12} = \frac{1}{4} [u_a + u_b + u^{(1)}_{22} + u^{(1)}_{11}] = \frac{1}{4} [10 + 40 + 10 + 7.5] = 16.875 \]

\[ u^{(2)}_{22} = \frac{1}{4} [u^{(1)}_{12} + u_b + u^{(1)}_{32} + u^{(1)}_{21}] = \frac{1}{4} [12.5 + 40 + 10 + 5] = 16.875 \]

\[ u^{(2)}_{32} = \frac{1}{4} [u^{(1)}_{22} + u_b + u_c + u^{(1)}_{31}] = \frac{1}{4} [10 + 40 + 0 + 5] = 13.75 \]
**Gauss-Seidel method:**

Consider the linear system $Ax = b$. One reason for slow convergence of Jacobi’s method is that it doesn’t make use of the latest information available. Gauss-Siedel method remedies this by using each new component of solution as soon as it has been computed.

Using the same notation as before, Gauss-Siedel method corresponds to splitting $M = D + L$, and $N = -U$ and can be written in matrix terms as:

$$Ax = b$$
$$A = D - L - U$$
$$((D - L) - U)x = b$$

$$(D - L)x^{\text{new}} = Ux^{\text{old}} + b$$

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

$$x^{(k+1)} = T_gx^{(k)} + C_g$$

where $T_g$ : is the Gauss-Seidel iteration matrix

$C_g$ : constant vector

If we apply Gauss-Siedel method to solve the system of finite difference equations for Poisson’s equation, we get:

$$U_j^{(k+1)} = \frac{1}{4}(U_{i-1,j}^{(k+1)} + U_{i,j-1}^{(k+1)} + U_{i+1,j}^{k} + U_{i,j+1}^{k} - h^2f(i,j))$$

(3.4)

Thus, we gain the average solution values at four surrounding grid points, the method uses the most recent values while Jacobi method uses values from the previous iteration. Gauss-Siedel method doesn’t always converge, but it is guaranteed to converge under conditions that are often
satisfied in practice, and are weaker than those for Jacobi’s method (e.g., if
matrix is symmetric and positive definite).

Although Gauss-Siedel converges more rapidly than Jacobi, it is still
not efficient for huge linear systems.

### 3.4 Successive Over - Relaxation:

Convergence rate of Gauss-Siedel can be accelerated by successive
over – relaxation (SOR), which in effect uses step to next Gauss-Siedel
iterate as search direction, but with a fixed search parameter denoted by
(\( \omega \)).

Starting with \( x^{(k)} \), first compute next iterate that would be given by
Gauss-Siedel, \( x_{gs}^{k+1} \) instead of taking next iterate to be:

\[
x^{K+1} = x^{(K)} + \omega(x_{gs}^{k+1} - x^{(K)})
\]

= \((1 - \omega)x^{(k)} + \omega x_{gs}^{k+1}\)

where \( \omega \) is fixed relaxation parameter chosen to accelerate convergence
which weighs average of current iterate and next Gauss-Siedel iterates.

\( \omega > 1 \) gives over- relaxation, and \( \omega < 1 \) gives under- relaxation and \( \omega = 1 \)
gives Gauss-Siedel method.

The method converges for \( 0 < \omega < 2 \) and diverges otherwise. In
general the optimal choice of \( \omega \) is not easy task and it is subject to theory
of special classes of matrices. If \( \rho_{jac} \) is the spectral radius of the Jacobi
iteration (so that its square is the spectral radius of the Gauss-Siedel
iteration), then the optimal choice for \( \omega \) is given by:

\[
\omega = \frac{2}{1 + \sqrt{1 - \rho_{jac}^2}}
\]
For this optimal choice, the spectral radius for (SOR) is

\[ \rho_{\text{SOR}} = \left( \frac{\rho_{\text{jac}}}{1 + \sqrt{1 - \rho_{\text{jac}}^2}} \right)^2 \]  

(3.7)

How do we choose \( \omega \) for a problem whose answer is not known analytically?

This is just the weak point of SOR! The advantages of SOR are obtained only in a fairly narrow window around the correct value of \( \omega \).

It is better to take \( \omega \) slightly too large, rather than slightly too small, but best to get the optimal value.

For reference purposes, we give the value of \( \rho_{\text{jac}} \) for our model problem on a rectangular \((nXm)\) grid, allowing the possibility that \( h \neq k \):

\[ \rho_{\text{jac}} = \cos \frac{\pi}{n} + \left( \frac{h}{k} \right)^2 \cos \frac{\pi}{m} \]

(3.8)

Using the same notation as before, SOR method corresponds to splitting

\[ M = \frac{1}{\omega} D + L, \quad N = \left( \frac{1}{\omega} - 1 \right) D - \omega \]

And can be written (see Smith, 1978; Mitchell, 1980) in matrix terms as:

\[ x^{(K+1)} = K^{(x)} + \omega(D^{-1}(b - Lx^{(K+1)} - Ux^{(K)}) - x^{(K)}) \]
\[ = (D + \omega L)^{-1}((1 - \omega)D - \omega U)x^{(K)} + \omega(D + \omega L)^{-1}b \]

If we apply SOR method to solve the system of finite difference equations for Laplace's equation, we get:

\[ \text{new}\ u(i, j) = (1 - \omega) \ast u(i, j) + 0.25 \ast \omega(u(i + 1, j) + \text{new}\ u(i - 1, j) + u(i, j + 1) + \text{new}\ u(i, j - 1)). \]
A necessary and sufficient condition for the convergence of iterative methods:

Each of three iterative methods described above can be written as:

$$x^{(k+1)} = Gx^k + c,$$

where $G$ is iteration matrix and $c$ a column vector of known values.

This equation was derived from the original equation by rearranging them into the form:

$$x = Gx + c$$

i.e., the unique solution of the $m$ linear equations $Ax = f$ is the solution of equation (3.11). The error $e^{(n)}$ in the $n$th approximation to the exact solution is defined by $e^{(n)} = x - x^{(n)}$ so it follows by the substraction of equation (3.10) from equation (3.11) that

$$e^{(n+1)} = Ge^{(n)}$$

Therefore

$$e^{(n)} = Ge^{(n-1)} = G^2e^{(n-2)} = ... = G^{(n)}e^{(0)}$$

(3.11)

The sequence of iterative values $x^{(1)}, x^{(2)}, ..., x^{(n)}, ...$ will converge to $x$ as $n$ tends to infinity if

$$\lim_{n \to \infty} G^{(n)} = 0.$$ 

Assume now that the matrix $G$ of order $m$ has $m$ linearly independent eigenvectors $\nu_s, s = 1(1)m$. then these eigenvectors can be used as a basis for our $m$-dimensional vector space and the arbitrary error vector $e^{(0)}$ with its
components, can be expressed uniquely as a linear combination of them, namely,

\[ e^{(1)} = \sum_{s=1}^{m} c_s v_s \]

where the \( c_s \) are scalars. Hence

\[ e^{(1)} = Ge^{(0)} = \sum_{s=1}^{m} c_s v_s . \]

But \( Gv_s = \lambda_s v_s \) by the definition of an eigenvalue, where \( \lambda_s \) is the eigenvalue corresponding to \( v_s \). Hence

\[ e^{(1)} = \sum_{s=1}^{m} c_s \lambda_s v_s . \]

Similarly,
\[ e^{(n)} = \sum_{s=1}^{m} c_s \lambda_s v_s . \]

Therefore \( e^{(n)} \) will tend to the null vector as \( n \) tends to infinity, for arbitrary \( e^{(0)} \), if and only if \(| \lambda_s | \leq 1\) for all \( s \). In other words, the iteration will converge for arbitrary \( x^{(0)} \) if and only if the spectral radius \( \rho(G) \) of \( G \) is less than one.

As a corollary to this result a sufficient condition for convergence is that \( \| G \| < 1 \). To prove this we have that \( Gv_s = \lambda_s v_s \). Hence

\[ \| Gv_s \| = \| \lambda_s v_s \| = | \lambda_s | \| v_s \| . \]

But for any matrix norm that is compatible with a vector norm \( \| v_s \| , \)

\[ \| Gv_s \| \leq \| G \| \| v_s \| . \]

Therefore \( \| \lambda_s \| \| v_s \| \leq \| G \| \| v_s \| , \)

So \( \| \lambda_s \| \leq \| G \| , \quad s = 1(1)m. \)
It follows from this that a sufficient condition for convergence is that \( \| G \| < 1 \). It is not a necessary condition because the norm of \( G \) can exceed one even when \( \rho(G) < 1 \).

3.5 Multigrid Method

Accuracy constraint on the numerical solutions of boundary value problems defined on large domains leads to huge systems of equations. Direct solvers are not efficient for such systems. Classical iterative methods also are not efficient. Multigrid methods are relatively new and they are very efficient. Multigrid methods accelerate iterative solvers. Historically, it was developed for elliptic problems which is mathematically well understood, but it is now used in many other situations.

Iterative methods start with an initial approximation for the solution to the differential equations. In each iteration, the difference between the approximate solution and the exact solution supposed to be smaller.

One can analyze this difference or error into components of different wavelength, for example, by using Fourier analysis. In general, the error will have components of many different wavelengths: there will be short wavelength error components and long wavelength error components.

Algorithms like Jacobi or Gauss-Seidel are local because the new value for the solution at any lattice site depends only on the value of the previous iterate at neighboring points. Such local algorithms are more efficient in reducing short wavelength error components.
The basic idea behind multigrid methods is to reduce long wavelength error components by updating blocks of grid points.

3.5.1 **Multigrid method for Poisson’s equation in 2-D:**

With small change in notation, poisson’s equation in 2-D can be written as:

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y),
\]

(3.12)

Where the unknown solution \( u(x, y) \) is determined by the given source term \( f(x, y) \) in a close region. Let’s consider a square domain \( 0 \leq x \leq 1 \) with homogenous Dirichlet boundary condition \( u = 0 \) on the perimeter of the square. The equation is discretized on a grid with \( L + 2 \) lattice points. i.e., \( L \) interior points and 2 boundary points, in the \( x \) and \( y \) directions. At any interior point, the approximation solution obeys

\[
u_{i,j} = \frac{1}{4}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - h^2 f_{i,j})
\]

(3.13)

The algorithm uses a succession of lattice or grid. The number of different grids is called the number of multigrid levels \( (l) \). The number of interior lattice points in the \( x \) and \( y \) directions is taken to be \( 2^l \), so that \( L = 2^l + 2 \), and the lattice spacing \( h = \frac{1}{(L-1)} \), \( L \) is close in this manner so that the downward multigrid iteration can construct a sequence of coarser lattice with

\[
2^{l-1} \rightarrow 2^{l-2} \rightarrow .... \rightarrow 2^0 = 1
\]

interior points in the \( x \) and \( y \) directions.
Suppose that \( u(x, y) \) is the approximate solution at any stage in the calculation, and \( u_{\text{exact}}(x, y) \) is the exact solution which we are trying to find, the multigrid algorithm uses the following definitions:

1) The correction

\[
\nu = u_{\text{exact}} - u
\]

(3.14)

is the function which must be added to the approximate solution to give the exact solution.

2) The residual or defect is defined as:

\[
r = \nabla^2 u + f
\]

(3.15)

Notice that the correction and the residual are related by the equation:

\[
\nabla^2 \nu = (\nabla^2 u_{\text{exact}} + f) - (\nabla^2 u + f) = -r
\]

(3.16)

This equation has exactly the same form as Poisson’s equation with \( \nu \), playing the role of unknown function and \( r \) playing the role of known source function.

### 3.5.2 Simple V-Cycle Algorithm

The simplest multigrid algorithm is based on a two-grid improvement scheme namely:

- A fine grid with \( L = 2^l + 2 \) points in each direction, and
- A coarse grid with \( L = 2^{l-1} + 2 \) points.
We need to be able to move from one grid to another, i.e., any given function on the lattice, we need to be able to:

• restrict the function from fine to coarse, and

• prolongate or interpolate the function from coarse to fine.

Given these definitions, the multigrid V-cycle can be defined recursively as follows:

• If \( l = 0 \) there is only one interior point, so solve exactly (or iteratively but exactly more accurate) for

\[
u_{i,j} = \left( u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - \frac{h^2}{4} f(i,j) \right) / 4.
\]

• Otherwise, calculate the current \( L = 2^l + 2 \)

• Perform a few pre-smoothing iterations using a local algorithm such as Gauss-Seidel. The idea is to damp or reduce the short wavelength errors in the solution.

• Estimate the correction \( v = u_{\text{exact}} - u \) as follows:

• Compute the residual
\[ r_{i,j} = \frac{1}{h^2} (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}) + f_{i,j}. \] 

(3.17)

- Restrict the residual \( r \rightarrow R \) to the coarser grid.
- Set the coarser grid correction \( V = 0 \) and improve it recursively.
- Prolongate the correction \( V \rightarrow \nu \) onto the finer grid.
- Correct \( u \rightarrow u + \nu \)
- Perform a few post-smoothing Gauss-Seidel (\( i \) ) iterations and return the improved \( u \).

How does this recursive algorithm scale with \( L \)? The pre-smoothing and post-smoothing Jacobi or Gauss-Seidel iterations are the most time consuming parts of the calculation. Recall that a single Jacobi or Gauss-Seidel iteration scales like \( O(L^2) \). The operations must be carried out on the sequence of grids with

\[
2^i \rightarrow 2^{i-1} \rightarrow 2^{i-2} \rightarrow \ldots \rightarrow 2^0 = 1
\]

interior lattice points in each direction. The total number of operations is of order

\[
L^2 \sum_{n=0}^{\log_2 L} \frac{1}{2^{2n}} \leq \frac{4}{3} L^2.
\]

Thus the multigrid V-cycle scales like \( O(L^2) \), i.e., linearly with the number of lattice points \( N \).

### 3.5.3 Restricting the Residual to a coarser Lattice

The coarser lattice with spacing \( H = 2h \) is constructed. A simple
algorithm for restricting the residual to the coarser lattice is to set its value to the average of the values on the four surrounding lattice points (cell-centered coarsening):

$$R_{i,j} = \frac{1}{4}(r_{i,j} + r_{i+1,j} + r_{i,j+1} + r_{i+1,j+1}), i = 2I - 1, j = 2J - 1.$$  \hspace{1cm} (3.18)

### 3.5.4 Prolongation of the Correction to the Fine Lattice

Having restricted the residual to the coarser lattice with spacing $H = 2h$, we need to solve the equation

$$\nabla^2 V = -R(x, y),$$

With the initial guess $V(x, y) = 0$. This is done by two-grid iteration

$$V = \text{twoGrid}(H, V, R).$$

The output must now be interpolated or prorogated to the finer lattice.

The simplest procedure is to copy the value of $V_{i,j}$ on the coarse lattice to the 4 neighboring cell points on the finer lattice:

$$V_{i,j} = V_{i+1,j} = V_{i+1,j+1} = V_{i,j+1} = V_{i+1,j+1}, i = 2I - 1, j = 2J - 1.$$  \hspace{1cm} (3.19)
Chapter Four

Computational Results
4.1 Introduction

In this chapter Laplace’s equation was treated, and it is solved by using some different methods.

Laplace’s equation in two dimensions can be written as:

\[ u_{xx} + u_{yy} = 0 \]

\[ S = \{(x, y) | 0 \leq x \leq a, 0 \leq y \leq b\} \]

\[ u(x, y) = g(x, y) \text{ on } \partial S \]

The above equation can be written after discretization as:

\[ u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - h^2 f(i,j)}{4} \] (4.1)

For example if we have a unit square \( S = \{(x, y) | 0 \leq x \leq 1, 0 \leq y \leq 1\} \) with Drichlet’s boundary condition

\[ U(0, y) = f_2, \quad U(1, y) = f_4. \]

\[ U(x,0) = f_3 \text{ and } U(x,1) = f_1. \]

4.2 Numerical Algorithm

Jacobi Method (JM):

Input: end points \( a, b, c, d \), integer \( m, n \), tolerance TOL, maximum number of iteration \( k \), \( f_1, f_2, f_3, f_4 \)
Output: Approximation $U_{i,j}$ for each $i = 1,\ldots,n-1$ and $j = 1,\ldots,m-1$

Make initial guess for $U^{(0)}_{i,j}$ at all interior points

Step 1  set $h = 1/(n)$,

$$x = (0 : n) * h,$$

$$y = (0 : n) * h,$$

Set $U_{i,j} = 0$,

Step 2  for $i = 1,\ldots, n$

Set $U(1,n) = feval(f_1, h*(i-1),0)$

Set $U(n,i) = feval(f_2, h*(i-1),0)$

Stop

Step 3  for $j = 1,\ldots, n$

Set $U(i,1) = feval(f_3, h*(i-1),0)$

Set $U(i,n) = feval(f_4, h*(i-1),0)$

Stop

Step 4  set $U_1 = U$

Set $k_{\text{max}} = n^3$

Set TOL=1e-4
Step 5 set for $k = 1, \ldots, k_{\max}$

Set frac_diff = 0

Step 6 set for $i = 2, \ldots, n - 1$

Set for $j = 2, n - 1$

Set $U_1 = U_{i,j}^{(k+1)} = \frac{1}{4}[U_{i+1,j}^k + U_{i-1,j}^k + U_{i,j+1}^k + U_{i,j-1}^k]$

Set frac_diff = frac_diff + abs $(U_i(i, j) - U(i, j))$

Stop

Stop

step 7 set change(k) = frac_diff/(n-2)^2

set if rem(k,100) < 1

Stop

Step 8 set if change (k) < TOL

Set break

Stop.

See the Matlab code in appendix (B.1)

**Gauss Seidel (GSM):**

Input: end points $a,b,c,d$ ,integer $m, and n$, tolerance TOL, maximum number of iteration(k), $f_1,f_2,f_3,f_4$
Output: Approximation $U_{i,j}$ for each $i = 1,\ldots,n-1$ and $j = 1,\ldots,m-1$

Make initial guess for $U^{(0)}_{i,j}$ at all interior points

Step 1 set $h = 1/(n)$,

\[ x = (0 : n) \ast h, \]

\[ y = (0 : n) \ast h, \]

set $U_{i,j} = 0$

step 2 for $i = 1,\ldots,n$

set $U(1,n) = feval(f_1, h \ast (i-1), 0)$

set $U(n,i) = feval(f_2, h \ast (i-1), 0)$

stop

step 3 for $j = 1,\ldots,n$

set $U(i,1) = feval(f_3, h \ast (i-1), 0)$

set $U(i,n) = feval(f_4, h \ast (i-1), 0)$

stop

step 4 set $U_i = U$

set $k_{\text{max}} = n^3$

set $\text{TOL} = 1e-4$
step 5 set for \( k = 1, \ldots, k_{\text{max}} \)

set frac_diff = 0

step 6 set for \( i = 2, \ldots, n - 1 \)

set for \( j = 2, n - 1 \)

set \( U_i = U_{i,j}^{(k+1)} = \frac{1}{4} [U_{i+1,j}^k + U_{i-1,j}^{(k+1)} + U_{i,j+1}^k + U_{i,j-1}^{(k+1)}] \)

set frac_diff = frac_diff + abs(\( U_i(i,j) - U(i,j) \))

stop

step 7 set change(k) = frac_diff/(n-2)^2

set if rem(k,100) < 1

stop

step 8 set if change(k) < TOL

set break

stop.

See the Matlab code in appendix (B.2)

**Successive Over-Relaxation Method (SORM)**

1. Make initial guess for \( U_{i,j} \) at all interior points (1,j)
2. Define a scalar \( w (0 < w < 2) \).

3. Apply equation (4.1) to all interior points \((i,j)\) and call it \( U'_{i,j} \).

4. \[ U_{i,j}^{(k+1)} = wU'_{i,j} + (1-w)U^k_{i,j} \]

5. Stop if prescribed convergence threshold is reached

6. \[ U_{i,j}^k = U^{k+1}_{i,j} \]

7. Go to step 2.

See the Matlab code in appendix (B.3)

**Multigrid Method (MGM):**

Input: charge\((q)\), the number of multigrid levels \( l \)

Set the number of interior lattice in \(x\) and \(y\) to be \(2^l\)

Set \( N = 2^l + 2 \)

Set \( h = \frac{1}{(N)} \)

Define \( u = \text{twogrid}(\text{level},h,u,f) \)

1. If \( l = 0 \), there is only one interior point, so solve exactly for:
   \[ u_{2,2} = (u_{1,2} + u_{3,2} + u_{2,1} + u_{2,3} + h^2f_{2,2})/4. \]

2. Calculate the current \( N = 2^l + 2 \)

3. Estimate the correction \( v = u_{\text{exact}} - u \) as follows:
   - Compute the residual \( r = \nabla^2 u + f \)
   - Restrict the residual \( r \to R \) to a coarse grid.
- Set the coarse grid correction \( V = 0 \).

- Improve the correction recursively \( V = \text{twogrid}(l-1,2h,V,R) \).

- Prolongate the correction \( V \rightarrow v \) onto the fine grid.

4. Correct \( u \rightarrow u + v \)

Set \( r=\text{zeros}(N,N) \)

Set for \( i=2,\ldots,N \)

Set for \( j=2,\ldots,N \)

\[
r(i,j) = (u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1)-4*u(i,j))/h^2+f(i,j)
\]

stop

set \( i=2(1-1), j=2(J-1) \)

\[
R_{i,j} = \frac{1}{4}[r_{i,j} + r_{i,j+1} + r_{i+1,j} + r_{i+1,j+1}]
\]

Set \( M = 2^{(\text{level}-1)} + 2 \)

Set \( R=\text{zeros}(M,M) \)

Set for \( I=2,\ldots,M \)

Set for \( J=2,\ldots,M \)Set \( R(I,J) = (r(i,j)+r(i+1,j)+r(1,j+1)+r(i+1,j+1))/4 \)

Stop

Stop.

See the Matlab code in appendix (B.4)
Now, let's begin with constant boundaries

\[ f_1 = 100, \quad f_2 = 50 \quad \text{and} \quad n = m = 10 \]

\[ f_3 = 300, \quad f_4 = 200. \]

Method (JM):
Method (GSM):
Method SORM:

Assume that the boundaries are function with respect to $x$ and $y$.

\[
\begin{align*}
  f_1 &= y^4, \\
  f_2 &= y^4 - 6y^2 + 1 \\
  f_3 &= x^4, \\
  f_4 &= x^4 - 6x^2 + 1
\end{align*}
\]

and $n = m = 10$. 

\[
\begin{align*}
  f_3 &= x^4, \\
  f_4 &= x^4 - 6x^2 + 1
\end{align*}
\]
Method JM:
Method GSM:
Method SORM:
If we use the rotated five points Stencil (RFPS)

For the same conditions we get:
Method MGM:

Multigrid Level 1: Pre-Smooth Iteration 8 difference between u and residual

Fractional Change

Iteration Number
If we use nine-points approximation for the above conditions, we obtain:
4.3 Poisson Equation

In this section Poisson's equation was treated.

- Consider Poisson's equation:

\[ u_{xx} + u_{yy} = -2 \]

\[ S = \{(x, y) | 0 \leq x \leq 2, 0 \leq y \leq 2\} \]

with boundary conditions as shown

![Diagram showing boundary conditions and grid]

The above equation can be written as:

\[ u(i, j) = \frac{1}{4}(u(i + 1, j) + u(i - 1, j) + u(i, j + 1) + u(i, j - 1)) + \frac{1}{2} \]

for node (1): \(1 + u_2 - 4u_1 + u_0 + \frac{1}{2} = 0\)

for node (2): \(1 + u_1 - 4u_2 + u_0 + \frac{1}{2} = 0\)
for node(3): \( u_s + u_2 - 4u_3 + u_4 + 0 + \frac{1}{2} = 0 \)

for node(4): \( u_s + u_3 - 4u_4 + 0 + 0 + \frac{1}{2} = 0 \)

for node(5): \( 1 + 1 - 4u_5 + u_5 + u_3 + \frac{1}{2} = 0 \)

The above equations can be written as a matrix form:

\[ A \mathbf{u} = \mathbf{b} \]

where

\[
A = \begin{bmatrix}
-4 & 2 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 \\
0 & 1 & -4 & 1 & 1 \\
0 & 0 & 2 & -4 & 0 \\
0 & 0 & 2 & 0 & -4 \\
\end{bmatrix},
\mathbf{b} = \begin{bmatrix}
-3 \\
2 \\
-1 \\
2 \\
-5 \\
\end{bmatrix},
\mathbf{u} = \begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
u_5 \\
\end{bmatrix}
\]

When the above work is done, we obtain:

\[
\mathbf{u} = \begin{bmatrix}
0.737 \\
0.723 \\
0.658 \\
0.454 \\
0.954 \\
\end{bmatrix}
\]

When numerical methods are used to solve the above problems we get:

The number of iteration for Jacobi method is \( 16 \)

\text{norm\_inf error for Jacobi method is } 0.0001680128

The solution by Jacobi method is:

\( 0.9538 \)
The number of iteration for Gauss Seidel method is = 9

norm_inf error for Gauss Seidel method is = 0.0001462412

The solution by Gauss Seidel method is:

0.9538
0.4538
0.6578
0.7236
0.7368

w = 1.1920

The number of iteration for SOR method is = 5

norm_inf error for SOR method is = 0.0001347322

The solution by SOR method is:

0.9539
0.4538
0.6580
The solution of $u(x,y)$ is

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.4539 & 0.6579 & 0.7237 & 0.7368 & 0.7237 & 0.6579 & 0.4539 & 0 \\
0 & 0.6579 & 0.9539 & 1.0000 & 1.0000 & 1.0000 & 0.9539 & 0.6579 & 0 \\
0 & 0.7237 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 0.7237 & 0 \\
0 & 0.7368 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 0.7368 & 0 \\
0 & 0.7237 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 0.7237 & 0 \\
0 & 0.6579 & 0.9539 & 1.0000 & 1.0000 & 1.0000 & 0.9539 & 0.6579 & 0 \\
0 & 0.4539 & 0.6579 & 0.7237 & 0.7368 & 0.7237 & 0.6579 & 0.4539 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- Helmholtz Equation

Consider Helmholtz equation:

\[
u_{xx} + u_{yy} - 12.5\pi^2 u = -25\pi^2 \sin(\frac{5\pi}{2} x) \sin(\frac{5\pi}{2} y)
\]

in regular region $s = \{x, y \mid 0 < x, y < 0.4\}$

subject to the Dirichlet boundary condition:

$u(x,y) = 0 \quad on \quad \partial s$
To compare numerical solution with exact solution, the exact is:

\[ u(x, y) = \sin\left(\frac{5}{2} \pi x\right) \sin\left(\frac{5}{2} \pi y\right) \]

The above equation can be written as:

\[
\begin{align*}
  u(i, j - 1) + u(i - 1, j) - (4 - 0.125\pi^2)u(i, j) + u(i + 1, j) + u(i, j + 1) \\
  &= -25\pi^2 \sin\left(\frac{5}{2} \pi x_i\right) \sin\left(\frac{5}{2} \pi y\right)
\end{align*}
\]

Let \( r = -4 - 0.125\pi^2 \)

We can write the above equation in a matrix form as:

\[ A \mathbf{u} = \mathbf{b} \]
where \( A = \begin{bmatrix} B & I & 0 \\ I & B & I \\ 0 & 0 & B \end{bmatrix} \), \( B = \begin{bmatrix} r & 1 & 0 \\ 1 & r & 1 \\ 0 & 1 & r \end{bmatrix} \)

\[
\begin{align*}
\mathbf{b} &= \begin{bmatrix}
-0.25\pi^2 \sin(2.5\pi x_1) \sin(2.5\pi y_1) \\
-25\pi^2 \sin(2.5\pi x_2) \sin(2.5\pi y_1) \\
-25\pi^2 \sin(2.5\pi x_3) \sin(2.5\pi y_1) \\
-25\pi^2 \sin(2.5\pi x_4) \sin(2.5\pi y_1) \\
-25\pi^2 \sin(2.5\pi x_5) \sin(2.5\pi y_2) \\
-25\pi^2 \sin(2.5\pi x_6) \sin(2.5\pi y_2) \\
-25\pi^2 \sin(2.5\pi x_7) \sin(2.5\pi y_2) \\
-25\pi^2 \sin(2.5\pi x_8) \sin(2.5\pi y_2) \\
-25\pi^2 \sin(2.5\pi x_9) \sin(2.5\pi y_3) \\
-25\pi^2 \sin(2.5\pi x_{10}) \sin(2.5\pi y_3) \\
\end{bmatrix}
\end{align*}
\]

see the Matlab code for this problem in appendix (B.9)

when the above system is solved, we get:

\[
\text{norm}_{\infty} \text{ error for Jacobi method is } = 0.00007
\]

The number of iteration for Jacobi method is = 33

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.1465 & 0.2706 & 0.3536 & 0.3827 & 0.3536 & 0.2706 & 0.1465 & 0 \\
0 & 0.2706 & 0.5000 & 0.6533 & 0.7072 & 0.6533 & 0.5000 & 0.2706 & 0 \\
0 & 0.3536 & 0.6533 & 0.8536 & 0.9239 & 0.8536 & 0.6533 & 0.3536 & 0 \\
\end{array}
\]
norm_inf error for Gauss Seidel method is = 0.00238

The number of iteration for Gauss Seidel method is = 17
\[ w = 1.4795 \]

\[ \text{norm inf error for sorl method is } = 0.00580 \]

The number of iteration for SOR method is = 10
If $h = k = 0.02$ we get:
Finally, consider the problem by using Neuman condition:

Let's solve Laplace's equation

\[ u_{xx} + u_{yy} = 0 \]

The boundaries shown in the figure below:

Laplace's equation can be written as:

\[ u(i, j - 1) + u(i - 1, j) - 4u(i, j) + u(i + 1, j) + u(i, j + 1) = 0 \]

For node (1): \(-4u_1 + u_2 + u_4 = -8\)

For node(2): \(u_1 - 4u_2 + u_3 + u_5 = -4\)

For node(3): \(u_2 - 4u_3 + C_1 + C_2 = -4\)

To find \(C_1\) and \(C_2\) by using central difference formula:

\[
\frac{\partial u}{\partial x} = \frac{C_1 - U_2}{2h} \Rightarrow C_1 = u_2 + 2h \frac{\partial u}{\partial x}
\]
\[
\frac{\partial u}{\partial y} = \frac{C_2 - 4}{2k} \quad \Rightarrow \quad C_2 = 4 + 2k \frac{\partial u}{\partial y}
\]

\[C_1 + C_2 = u_2 + 2h \frac{\partial u}{\partial x} + 4 + 2k \frac{\partial u}{\partial y}\]

for: \( h = k = 0.1 \)

\[= u_2 + 2h(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}) + 4 \]

but \( \frac{\partial u}{\partial n} = \frac{x + y}{(2)^{1/2}} \)

\[= \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} \]

\[= u_2 + 2h(\frac{x + y}{(2)^{1/2}}) + 4 \]

\[= u_2 + 2(1)(\frac{3 + 1}{(2)^{1/2}}) + 4 \]

\[= u_2 + 4.0565685 \]

\[u_2 - 4u_4 + u_2 + 4.0565685 = -4 \]

\[-4u_3 + 2u_2 = -8.0565685 \]

for node (4): \( u_1 - 4u_4 + u_5 + C_1 = -4 \)

to find \( C_1 \), we use the same procedure then:

\[C_1 = u_1 + 0.02 \]

\[2u_1 - 4u_4 + u_5 = -4.02 \]

for node (5): \( u_2 + u_4 - 4u_5 + C_1 + C_2 = 0 \)

\[C_1 + C_2 = u_4 + u_2 + 0.0565685 \]
\( u_2 + u_4 + -4u_3 + u_4 + u_2 = -0.0565685 \)

The above equations can be written as a matrix form:

\[
Au = b
\]

where

\[
A = \begin{bmatrix}
-4 & 1 & 0 & 1 & 0 \\
1 & -4 & 1 & 0 & 1 \\
0 & 2 & -4 & 0 & 0 \\
2 & 0 & 0 & -4 & 1 \\
0 & 2 & 0 & 2 & -4
\end{bmatrix},
\]

\[
b = \begin{bmatrix}
-8 \\
-4 \\
-8.0565685 \\
-4.02 \\
-0.0565685
\end{bmatrix}
\]

see the Matlab code in appendix \textbf{(B.10)}

If we solve the above system by using numerical methods we obtain:

The solution by matrix

4.0078

4.0147

4.0215

4.0163

4.0297

\begin{center}
\begin{tabular}{c}
\hline
The number of iteration for jacobi method is=100 \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c}
\hline
norm of error for jacobi method is=0.0000000001 \\
\hline
\end{tabular}
\end{center}

The solution by Jacobi's Method is

4.0078
the number of iteration for Gauss Seidel method is=50

norm of error for Gauss Seidel method is=0.0000000002

The solution by Gauss Seidel Method is

4.0078
4.0147
4.0215
4.0163
4.0297

ω =1.5279

The number of iteration for SOR method is=35

norm of error for SOR method is=0.0000000005

The solution by SOR Method is

4.0078
4.0147
The solution of \( u(x,y) \) is:

\[
\begin{matrix}
4.0000 & 4.0163 & 4.0297 & 0 & 0 \\
4.0000 & 4.0078 & 4.0147 & 4.0215 & 0 \\
4.0000 & 4.0000 & 4.0000 & 4.0000 & 4.0000 \\
\end{matrix}
\]

4.4 Laplace's equation in Polar Coordinate

Now, let's deal with Laplace equation in a polar coordinate

\[
\nabla^2 u = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} = 0
\]

Let \( i \) and \( j \) be the index in the \( r \) and \( \theta \) directions respectively, the finite difference form of the Laplace equation is written as:

\[
\nabla^2 u = \frac{u(i-1, j) - 2u(i, j) + u(i+1, j)}{\Delta r^2} + \frac{1}{r_i} \frac{u(i+1, j) - u(i-1, j)}{2\Delta r} + \frac{1}{r_i^2} \frac{u(i, j-1) - 2u(i, j) + u(i, j+1)}{\Delta \theta^2} = 0
\]

The value at the node \((i, j)\) can be computed from
\[
2 \left[ \frac{1}{\Delta r^2} + \frac{1}{r_i^2 \Delta \theta} \right] u(i,j) = \frac{u(i-1, j) + u(i+1, j)}{\Delta r^2} + \frac{1}{r_i} \frac{u(i+1, j) - u(i-1, j)}{2 \Delta r} + \frac{1}{r_i^2} \frac{u(i, j-1) + u(i, j+1)}{\Delta \theta^2}
\]

A semicircular plate with a radius of 1 has the straight boundary held at 0 while the curve boundary is held at 100. Find \( u(r, \theta) \) with the grid given in Figure

The nodes in a two dimensional, polar coordinate system.

See the Matlab code for this problem in appendix (B.11)

# Of iteration =46

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
2.4986e+00 & 2.3284e+00 & 1.8232e+00 & 1.0139e+00 & 0 \\
4.8038e+00 & 4.5534e+00 & 3.7450e+00 & 2.2370e+00 & 0
\end{array}
\]
83

6.8366e+001  6.6121e+001  5.8055e+001  3.9163e+001  0

8.5659e+001  8.4422e+001  7.9457e+001  6.3678e+001  0

1.0000e+002  1.0000e+002  1.0000e+002  1.0000e+002  1.0000e+002
4.5 Conclusion and Results

Various numerical methods namely: Jacobi, Gauss-Seidel, SOR, Multigrid have been studied in order to compare the efficiency of these methods. It is desirable to develop the rapid method to obtain the accurate solutions, since the systems of equations are becoming large and many computer resources are needed.

From the above work, we see that multigrid method is the fastest. On other hand, the running time for multigrid methods is of order three (i.e. $O(n^3)$), while the other methods is of order five (i.e. $O(n^5)$).

So, multigrid method is superior to other methods, because it speeds up the convergence of relaxation method. Multigrid method is known as the procedure which takes only $O(n)$ operations.
Appendix

Programs

This part contains some programs used to solve Laplace's equation in different conditions

B.1

Parent routine for Method JM in case one

% Programme to solve Laplace's equation on the unit square
% using the Jacobi's method

Clear; help Jacobi;

n = input('Enter the number of grid points on each side-');

h = 1/(n - 1);

x = (0:n-1)*h;

y = (0:n-1)*h;

u = zeros(n);

u_x0 = input('Enter the potential at x=0 -');

u(1,:) = u_x0;

u_x1 = input('Enter the potential at x=1 -');

u(n,:) = u_x1;

u_y0 = input('Enter the potential at y=0 -');

u_y1 = input('Enter the potential at y=1 -');

for i = 1:n
    for j = 1:n
        u(i,j) = 1/4 * (u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1));
    end
end

disp('Solution computed using Jacobi Method');

% Display the final solution grid
disp('x-values:'); disp(x);
disp('y-values:'); disp(y);
disp('Potential values:'); disp(u);
\texttt{u(:,1) = u\_y0;} \\
\texttt{u\_y1 = input('Enter the potential at y=1 -');} \\
\texttt{u(:,n) =u\_y1;} \\
\texttt{u1 = u;} \\
\texttt{k\_max = n^3;} \\
\texttt{tolerance = 1.e-04;} \\
\texttt{for k = 1:k\_max} \\
\hspace{1em} \texttt{frac\_diff = 0;} \\
\hspace{1em} \texttt{for i = 2:n-1} \\
\hspace{2em} \texttt{for j = 2:n-1} \\
\hspace{3em} \texttt{u1(i,j) = 0.25*(u(i+1,j) + u(i-1,j) +u(i,j+1) +u(i,j-1));} \\
\hspace{3em} \texttt{frac\_diff = frac\_diff + abs(u1(i,j)-u(i,j));} \\
\hspace{2em} \texttt{end} \\
\hspace{1em} \texttt{end} \\
\hspace{1em} \texttt{u = u1;} \\
\hspace{1em} \texttt{change(k) = frac\_diff/(n-2)^2;} \\
\hspace{1em} \texttt{if rem(k,100) < 1} \\
\hspace{2em} \texttt{fprintf('Fractional difference is %g after %g steps\n',change(k),k);}
end

if change(k) < tolerance

    break;

end

end

u

mesh(u);

pause;

semilogy(change);

xlabel('Iteration Number'); ylabel('Fractional Change');

B .2 Parent routine for Method GSM case one

% gseidel - Programme to solve Laplace's equation on the unit square

% using the Gauss Seidel method

clear; help gseidel;

n = input('Enter the number of grid points on each side - ');

h = 1/(n - 1);

x = (0:n-1)*h;
\[
y = (0:n-1)*h;
\]
\[
u = zeros(n);
\]
\[
u_x0 = input('Enter the potential at x=0 -');
\]
\[
u(1,:) = phi_x0;
\]
\[
u_x1 = input('Enter the potential at x=1 -');
\]
\[
u(n,:) = u_x1;
\]
\[
u_y0 = input('Enter the potential at y=0 -');
\]
\[
u(:,1) = u_y0;
\]
\[
u_y1 = input('Enter the potential at y=1 -');
\]
\[
u(:,n) = u_y1;
\]
\[
newu = u;
\]
\[
k_max = n^3;
\]
\[
tolerance = 1.e-04;
\]
\[
for k = 1:k_max
\]
\[
    frac_diff = 0;
\]
\[
    for i = 2:n-1
\]
\[
        for j = 2:n-1
\]
\[
            newu(i,j) = 0.25*(u(i+1,j) + newu(i-1,j) +u(i,j+1) +
\]
\[
                newu(i,j-1));
\]
frac_diff = frac_diff + abs(newu(i,j)-u(i,j));

end

end

u = newu;

change(k) = frac_diff/(n-2)^2;

if rem(k,100) < 1
    fprintf('Fractional difference is %g after %g steps
',change(k),k);
end

if change(k) < tolerance
    break;
end

end

u

mesh(u);

pause;

semilogy(change);

xlabel('Iteration Number');ylabel('Fractional Change');
B .3 Parent routine for Method SORM  case one

% sor - Programme to solve Laplace's equation on the unit square

% using the Successive Overrelaxation method

format ;

clear; help sor;

n = input('Enter the number of grid points on each side - ');

h = 1/(n - 1);

x = (0:n-1)*h;

y = (0:n-1)*h;

omega = 2/(1 + sin(pi/n));

u = zeros(n);

u_x0 = input('Enter the potential at x=0 - ');

u(1,:) = u_x0;

u_x1 = input('Enter the potential at x=1 - ');

u(n,:) = u_x1;

u_y0 = input('Enter the potential at y=0 - ');

u(:,1) = u_y0;
u_y1 = input('Enter the potential at y=1 -');

u(:,n) = u_y1;

newu = u;

k_max = n^3;

tolerance = 1.e-04;

for k = 1:k_max

    frac_diff = 0;

    for i = 2:n-1

        for j = 2:n-1

            newu(i,j) = (1 - omega)*u(i,j) + 0.25*omega*(u(i+1,j) + newu(i-1,j) + u(i,j+1) + newu(i,j-1));

            frac_diff = frac_diff + abs(newu(i,j)-u(i,j));

        end

    end

    u= newu;

    change(k) = frac_diff/(n-2)^2;

    if rem(k,100) < 1

        fprintf('Fractional difference is %g after %g steps\n',change(k),k);

    end

end
if change(k) < tolerance
    break;
end

end

u

mesh(u);
pause;

semilogy(change);

xlabel('Iteration Number'); ylabel('Fractional Change');

B .4 Parent routine for Method JM case two

% sor - Programme to solve Laplace's equation on the unit square

% using the Successive Overrelaxation method

format ;

clear; help sor;

n = input('Enter the number of grid points on each side - ');

h = 1/(n - 1);
x = (0:n-1)*h;

y = (0:n-1)*h;

omega = 2/(1 + sin(pi/n));

u = zeros(n);

u_x0 = input('Enter the potential at x=0 - ');

u(1,:) = u_x0;

u_x1 = input('Enter the potential at x=1 - ');

u(n,:) = u_x1;

u_y0 = input('Enter the potential at y=0 - ');

u(:,1) = u_y0;

u_y1 = input('Enter the potential at y=1 - ');

u(:,n) = u_y1;

newu = u;

k_max = n^3;

tolerance = 1.e-04;

for k = 1:k_max
    frac_diff = 0;
    for i = 2:n-1
        for j = 2:n-1
...
newu(i,j) = (1 - omega)*u(i,j) + 0.25*omega*(u(i+1,j)+ newu(i-1,j)+u(i,j+1) + newu(i,j-1));
frac_diff = frac_diff + abs(newu(i,j)-u(i,j));
end
end
u = newu;
change(k) = frac_diff/(n-2)^2;
if rem(k,100) < 1
    fprintf('Fractional difference is %g after %g steps\n',change(k),k);
end
if change(k) < tolerance
    break;
end
end
u
mesh(u);
pause;
semilogy(change);
B.5 Parent routine for Method GSM case two

% gseidel - Programme to solve Laplace's equation on the unit square

% using the Gauss Seidel method

format;

clear; help gseidel;

n= input('Enter the number of grid points on each side -');

f1 = inline('y^4+0*x');

f2 = inline('y^4-6*y^2+1+0*x');

f3 = inline('0*y+x^4');

f4 = inline('0*y+x^4-6*x^2+1');

h = 1/(n - 1);

x = (0:n-1)*h;

y = (0:n-1)*h;

u = zeros(n);

for i=1:n
    u1(1,i)=feval(f1,h*(i-1),0);
\[ u_1(n,i) = \text{feval}(f2, h \times (i-1), 0); \]

end

for \( i = 1:n \)

\[ u_1(i,1) = \text{feval}(f3, h \times (i-1), 0); \]

\[ u_1(i,n) = \text{feval}(f4, h \times (i-1), 0); \]

end

\[ k_{\text{max}} = n^3; \]

tolerance = 1.e-04;

for \( k = 1:k_{\text{max}} \)

\[ \text{changesum} = 0; \]

for \( i = 2:n-1 \)

\[ \text{for } j = 2:n-1 \]

\[ \% u_1(i,j) = 0.25 \times (u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1)); \]

\[ u_1(i,j) = 0.25 \times (u_1(i-1,j-1) + u_1(i+1,j-1) + u_1(i+1,j+1) + u_1(i-1,j+1)); \]

\[ \text{changesum} = \text{changesum} + \text{abs}(u_1(i,j) - u(i,j)); \]

end

end
\begin{verbatim}

   u = u1;

   change(k) = changesum / (n-2)^2;

   if rem(k,100) < 1

       fprintf('Error is %g after %g steps\n',change(k),k);

   end

   if change(k) < tolerance

       break;

   end

end

u

mesh(u);

pause;

semilogy(change);

xlabel('Iteration Number');ylabel('Error');

\end{verbatim}

\textbf{B .6} Parent routine for Method SORM case two

% sor - Programme to solve Laplace's equation on the unit square
% using the Successive Overrelaxation method

format ;

clear; help sor;

n= input('Enter the number of grid points on each side - ');

f1 = inline('y^4+0*x');

f2 = inline('y^4-6*y^2+1+0*x');

f3 = inline('0*y+x^4');

f4 = inline('0*y+x^4-6*x^2+1');

h = 1/(n - 1);

x = (0:n-1)*h;

y = (0:n-1)*h;

omega = 2/(1 + sin(pi/n));

u = zeros(n);

for i=1:n
    u(1,i)=feval(f1,h*(i-1),0);
    u(n,i)=feval(f2,h*(i-1),0);
end

for i=1:n
\begin{verbatim}
u(i,1)=feval(f3,h*(i-1),0);

u(i,n)=feval(f4,h*(i-1),0);

end

newu = u;

k_max = n^3;

tolerance = 1.e-04;

for k = 1:k_max
    frac_diff = 0;
    for i = 2:n-1
        for j = 2:n-1
            newu(i,j) = (1 - omega)*u(i,j) + 
            0.25*omega*(u(i,j+1)+newu(i-1,j) + u(i-1,j) + newu(i,j-1));
            frac_diff = frac_diff + abs(newu(i,j)-u(i,j));
        end
    end
    u= newu;
    change(k) = frac_diff/(n-2)^2;
end
\end{verbatim}
if rem(k,100) < 1

    fprintf('Fractional difference is %g after %g steps\n',change(k),k);

end

if change(k) < tolerance

    break;

end

end

u

mesh(u);

pause;

semilogy(change);

xlabel('Iteration Number');ylabel('Fractional Change');

B.7

function multigrid

% multigrid.m

% solves the two dimensional Poisson equation
% d^2phi/dx^2 + d^2phi/dy^2 = - rho(x,y)
% phi = 0 on the boundaries of a unit square
% using a simple multigrid method

clear all; help multigrid;

f1 = inline('yy^4+0*xx');

f2 = inline('yy^4-6*yy^2+1+0*xx');

f3 = inline('0*yy+xx^4');

f4 = inline('0*yy+xx^4-6*xx^2+1');

levels = input('Enter number of multigrid levels: ');

N = 2^levels + 2;                        % lattice points in 
x and y

disp(sprintf('Using a %i x %i square lattice', N, N));

h = 1/(N-1);                            % lattice spacing

% set charge density rho(x,y)

rho = zeros(N,N);

nq = 1;

while 1
disp(sprintf('To add charge number %i at lattice site ', nq));

q = input('Enter charge q or hit RETURN to continue: ');
if isempty(q), break, end

user_entry = input('Enter lattice coordinates [x y]: ');
if length(user_entry) == 2
    x = user_entry(1);
    y = user_entry(2);
end

rho(x,y) = q / h^2;

nq = nq + 1;
end

phi = zeros(N,N); % potential array

n=N;

h = 1/(n - 1);

xx = (0:n-1)*h;

yy = (0:n-1)*h;

for i=1:n
    phi(1,i)=feval(f1, h*(i-1), 0);
\[
\text{phi}(n,i) = \text{feval}(f2, h^{(i-1)}, 0);
\]
end

for \( i = 1 : n \)

\[
\text{phi}(i,1) = \text{feval}(f3, h^{(i-1)}, 0);
\]
\[
\text{phi}(i,n) = \text{feval}(f4, h^{(i-1)}, 0);
\]
end

figure;

[phi] = twogrid(levels, h, phi, rho); % start multigrid iteration

plotphi(phi, sprintf('Multigrid Iteration With %i Levels Done', levels));

uex = exact(N,h);

% figure;

% plotphi(uex, sprintf('Multigrid Iteration exact'));

function [u] = twogrid (level, h, u, f)

uin = u;

% recursive implementation of the simple multigrid algorithm

if level == 0 % solve exactly

\[
u(2,2) = 0.25 \times (u(3,2) + u(1,2) + u(2,3) + u(2,1)) \ldots
\]
+ h^2 * f(2,2));

return

end

N = 2^level + 2;

ngs = 3; % number of Gauss-Seidel steps

for n = 1:ngs % pre-smoothing GS steps
    u = GaussSeidel(N, h, u, f);
    msg = sprintf('Multigrid Level %i: Pre-Smooth Iteration %i', level, n);
    figure;
    plotphi(u, msg);
    r = zeros(N,N);
    for i = 2:N-1
        for j = 2:N-1
            r(i,j) = (u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) - 4 * u(i,j)) / h^2 + f(i,j);
        end
    end
end
figure;

msg = sprintf('Multigrid Level %i: Pre-Smooth Iteration %i difference between u and residual', level, n);

plotphi(u-r,msg);

% pause

msg = sprintf('residual after %i iteration and %i level is r', n, level);

disp(msg);

r

end

% find the residual

% restrict residual to coarser grid

M = 2^(level-1) + 2;

R = zeros(M,M);

for I = 2:M-1

    for J = 2:M-1

        i = 2*(I-1);

        j = 2*(J-1);

        R(I,J) = (r(i,j) + r(i+1,j) + r(i,j+1) + r(i+1,j+1)) / 4;

end
end

end

% set V on coarse grid to zero and call twogrid recursively

V = zeros(M,M);

[V] = twogrid(level-1, 2*h, V, R);

% prolongate V to the finer grid using simple injection

v = zeros(N,N);

for I = 2:M-1
    for J = 2:M-1
        i = 2*(I-1);
        j = 2*(J-1);
        v(i,j) = V(I,J);
        v(i+1,j) = V(I,J);
        v(i,j+1) = V(I,J);
        v(i+1,j+1) = V(I,J);
    end
end

% apply correction to u
for i = 2:N-1
    for j = 2:N-1
        u(i,j) = u(i,j) + v(i,j);
    end
end

for n = 1:ngs
    % post-smoothing GS steps
    u = GaussSeidel(N, h, u, f);
    msg = sprintf('Multigrid Level %i: Post-Smooth Iteration %i', level, n);
    figure;
    plotphi(u, msg);
end

function u = GaussSeidel (N, h, u, f)
    % Gauss-Seidel relaxation step with checkerboard updating
    for color = 0:1
        % red lattice : black lattice
        for i = 2:N-1
            for j = 2:N-1
                if mod(i+j,2) == color
                    u(i,j) = u(i,j) + v(i,j);
                end
            end
        end
    end
\[ u(i,j) = 0.25 \times (u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1)) + h^2 \times f(i,j); \]
N=4;
L1=2;
L2=2;
Uin=1;
Uot=0;

dy=L2/N;

dx=L1/N;

U=ones(2*N+1,2*N+1);

A=[-4 0 2 0 0; 0 -4 2 0 0; 1 1 -4 1 0; 0 0 1 -4 1; 0 0 0 2 -4];
b=[-5/2;-1/2;-1/2;-3/2;-3/2];

X=inv(A)*b;

disp('the solution by inverse matrix')
disp(X);
disp('---------------------------------')

for i=1:1:2*N+1
    U(1,i)=0;
    U(2*N+1,i)=0;
    U(i,1)=0;
    U(i,2*N+1)=0;
end
for i=1:1:3
    U(2,2*N+1-i)=X(i+1,1);
    U(2*N,2*N+1-i)=X(i+1,1);
end

for i=3:1:5
    U(2,i)=X(i,1);
    U(2*N,i)=X(i,1);
    U(i,2)=X(i,1);
    U(2*N+2-i,2)=X(i,1);
end

for i=2:1:4
    U(2*N,2*N+2-i)=X(i,1);
    U(i,2*N)=X(i,1);
end

for i=3:1:5
    U(i,2*N)=X(i,1);
    U(2*N+2-i,2*N)=X(i,1);
end
for i=2:1:3
    U(i,i)=X(N-i,1);
    U(2*N+2-i,i)=X(N-i,1);
    U(i,2*N+2-i)=X(N-i,1);
    U(2*N+2-i,2*N+2-i)=X(N-i,1)
end

L=zeros(N+1,N+1);
UU=zeros(N+1,N+1);
D=zeros(N+1,N+1);

for i=1:1:N+1
    for j=1:1:N+1
        if i==j
            D(i,j)=A(i,j);
        elseif i<j
            UU(i,j)=A(i,j);
        elseif i>j
            L(i,j)=A(i,j);
        end
    end
end
%-----------------Jacobi Method----------------------

Xjc=ones(N+1,1);
Xnew=zeros(N+1,1);
c=0;

while c~=15
    c=c+1;
    Xnew(1,1)=(5/2+2*Xjc(3,1))/4;
    Xnew(2,1)=(1/2+2*Xjc(3,1))/4;
    Xnew(3,1)=(1/2+Xjc(1,1)+Xjc(2,1)+Xjc(4,1))/4;
    Xnew(4,1)=(3/2+Xjc(3,1)+Xjc(5,1))/4;
    Xnew(5,1)=(3/2+2*Xjc(4,1))/4;
    Xjc=Xnew;
end

errjc=norm(Xjc(:)-X(:),inf); %norm error

disp(sprintf('number of iteration for Jacobi method is=%.0f',c))

disp(sprintf('norm_inf error for Jacobi method is=%.10f',errjc))
disp('The solution by Jacobi method is')

disp(Xjc)

disp('--------------------------------------------')

%-------------Jacobi method by LUD matrix-----

c=0;

Xoldjc=zeros(N+1,1);

Xjcm=zeros(N+1,1);

while c~=16

    c=c+1;

    Xjcm=(inv(D)*b)-((inv(D))*((L+UU)*Xoldjc));

    Xoldjc=Xjcm;

end

errjcm=norm(Xjcm(:)-X(:),inf); %norm error

disp(sprintf('number of iteration for Jacobi method is=%.0f',c))

disp(sprintf('norm_inf error for Jacobi method is=%.10f',errjcm))

disp('The solution by Jacobi method is')

disp(Xjcm)
%--------------------Gauss Seidel Method---------------

Xgs=ones(N+1,1);

C=0;

while c~=8
    c=c+1;

    Xgs(1,1)=(5/2+2*Xgs(3,1))/4;
    Xgs(2,1)=(1/2+2*Xgs(3,1))/4;
    Xgs(3,1)=(1/2+Xgs(1,1)+Xgs(2,1)+Xgs(4,1))/4;
    Xgs(4,1)=(3/2+Xgs(3,1)+Xgs(5,1))/4;
    Xgs(5,1)=(3/2+2*Xgs(4,1))/4;
end

errgs=norm(Xgs(:)-X(:),inf);  %norm error

disp(sprintf('number of iteration for Gauss Seidel method is=%.0f',c))

disp(sprintf('norm_inf error for Gauss Seidel method is=%.10f',errgs))

disp('The solution by Gauss Seidel method is')

disp(Xgs)
c=0;

Xoldgs=zeros(N+1,1);

Xgsm=zeros(N+1,1);

while c~=9
    c=c+1;
    Xgsm=(inv(L+D)*b)-((inv(L+D))*UU*Xoldgs);
    Xoldgs=Xgsm;
end

errgsm=norm(Xgsm(:)-X(:),inf); %norm error

disp(sprintf('number of iteration for Gauss Seidel method is=%.0f',c))

disp(sprintf('norm_inf error for Gauss Seidel method is=%.10f',errgsm))

disp('The solution by Gauss Seidel method is')

disp(Xgsm)

disp('--------------------------------------------')

%----------------------SOR Method-------------------
Xsor=ones(N+1,1);

c=0;

w=2/(1+sin(pi*dx))+.192

while c~<5

c=c+1;

Xsor(1,1)=(1-w)*Xsor(1,1)+(w*(5/2+2*Xsor(3,1)))/4;

Xsor(2,1)=(1-w)*Xsor(2,1)+(w*(1/2+2*Xsor(3,1)))/4;

Xsor(3,1)=(1-w)*Xsor(3,1)+(w*(1/2+Xsor(1,1)+Xsor(2,1)+Xsor(4,1)))/4;

Xsor(4,1)=(1-w)*Xsor(4,1)+(w*(3/2+Xsor(3,1)+Xsor(5,1)))/4;

Xsor(5,1)=(1-w)*Xsor(5,1)+(w*(3/2+2*Xsor(4,1)))/4;

end

errsor=norm(Xsor(:)-X(:),inf); %norm error

disp(sprintf('number of iteration for SOR method is=%.0f',c))

disp(sprintf('norm_inf error for SOR method is=%.10f',errsor))

disp('The solution by SOR method is')

disp(Xsor)
disp('--------------------------------------------------------')

disp('The solution of U(x,y) is')

disp(U)

%**********************************************************

B.9 Matlab code for Helmholtz equation/Chapter Four

    format short;

    h=.05;

    k=.05;

    L1=.4;

    L2=.4;

    Ux0=0;

    UxL=0;

    U0y=0;

    ULy=0;

    n=L1/h;

    U=zeros(n+1,n+1);

    for i=1:1:n+1

        U(1,i)=Ux0;

        U(n,i)=UxL;
U(i,1)=U0y;

U(i,n)=ULy;

end

a=n-1; % the number of unknown elements

x=h;

xy=zeros(a,1);

for i=1:1:a

xy(i,1)=x;

x=x+h;

end

A=zeros(a*a,a*a);

for i=1:1:a*a

A(i,i)=-4+(-12.5*(pi^2)*(h^2));

end

for i=1:1:(n-2)*a

A(i,i+a)=1;

A(a+i,i)=1;

end

for i=1:1:a-1
\begin{verbatim}
A(i,i+1)=1;
A(i+1,i)=1;
end
for m=1:1:a-2
  for N=1:1:a-2
    if m==N
      for i=1:1:a-1
        A(m*a+i,m*a+i+1)=1;
        A(m*a+i+1,m*a+i)=1;
      end
    end
  end
end
for i=1:1:a-1
  A((m-2)*a+i,(m-2)*a+i+1)=1;
  A((m-2)*a+i+1,(m-2)*a+i)=1;
end
A;
L=zeros(a*a,a*a);
\end{verbatim}
UU=zeros(a*a,a*a);
D=zeros(a*a,a*a);
for i=1:1:a*a
    for j=1:1:a*a
        if i==j
            D(i,j)=A(i,j);
        elseif i<j
            UU(i,j)=A(i,j);
        elseif i>j
            L(i,j)=A(i,j);
        end
    end
end
b=zeros(a*a,1);
for i=1:1:n-1
    b(i,1)=(-25*(pi^2)*sin(2.5*(pi)*xy(i,1))*sin(2.5*(pi)*xy(1,1))*(h^2)-U(1,i+1);
end
for m=1:1:a-2
    for i=1:1:n-1
        b(m*a+i,1)=(-25*(pi^2)*sin(2.5*(pi)*xy(i,1))*sin(2.5*(pi)*xy(m+1,1)))*(h^2);
    end
end

for i=1:1:n-1
    b(a*a-a+i,1)=(-25*(pi^2)*sin(2.5*(pi)*xy(i,1))*sin(2.5*(pi)*xy(1,1)))*(h^2)-U(n,i+1);
end

X=inv(A)*b;

r=0;

for i=2:1:n
    for j=2:1:n
        U(i,j)=X(r+(j-1),1);
    end
    r=r+n-1;
end
Uex=zeros(n+1,n+1);
y=0;
for i=1:1:n+1
    x=0;
    for j=1:1:n+1
        Uex(i,j)=sin(2.5*pi*x)*sin(2.5*pi*y);
        x=x+h;
    end
    y=y+k;
end
err_iter_inv=norm(U(:)-Uex(:),inf)
disp('The exact solution is')
disp(Uex)
disp(sprintf('norm_inf error for invers matrix method is ' +%5f',err_iter_inv))
disp('The solution by invers matrix is')
disp(U)
disp('-----------------------------------')
%---------------------Jacobi method-------------------
Ujc=zeros(n+1,n+1);
c=0;
R=4+((12.5*(pi^2)*(h^2)));

while c~=33
    c=c+1;
    k_max = (n)^3;
    tolerance = 1.e-04;

    for k = 1:k_max
        frac_diff = 0;

        for i=2:a+1
            for j=2:a+1
                Unew(i,j)=(((25*(pi^2)*sin(2.5*pi*xy(j-1,1))*sin(2.5*pi*xy(i-1,1)))*(h^2))+Ujc(i-1,j)+Ujc(i,j-1)+Ujc(i,j+1)+Ujc(i+1,j))/R;

                frac_diff=frac_diff+abs(Ujc(i,j)-Uex(i,j));
            end
        end

        Ujc(2:a+1,2:a+1)=Unew(2:a+1,2:a+1);

        change(k) = frac_diff/(n-2)^2;
if rem(k,100) < 1

    fprintf('Fractional difference is %g after %g steps\n',change(k),k);

end

if change(k) < tolerance

    break;

end

end

Ujc

mesh(Ujc);

pause;

semilogy(change);

xlabel('Iteration Number');ylabel('Fractional Changejc');

% err_iter_jc=norm(Ujc(:)-Uex(:,inf);

%end

%disp(sprintf('norm_inf error for jacobi method is =%.5f',err_iter_jc))

%disp(sprintf('number of iteration for jacobi method is =%.0f',c))

%disp(Ujc)
% disp('---------------------------------')

% ------------------- jacobi method LUD matrix -----------

c=0;

Ujcm=zeros(n+1,n+1);

Xoldjc=zeros(a*a,1);

xjcm=zeros(a*a,1);

while c~=33
    c=c+1;

    Xjcm=(inv(D)*b)-((inv(D))*((L+UU)*Xoldjc));

    Xoldjc=Xjcm;

end

r=0;

for i=2:1:n
    for j=2:1:n
        Ujcm(i,j)=Xjcm(r+(j-1),1);
    end
    r=r+n-1;
end
err_iter_jcm=norm(Ujcm(:)-Uex(:),inf);
disp(sprintf('norm_inf error for jacobi method is =%.5f',err_iter_jcm))
disp(sprintf('number of iteration for jacobi method is =%.0f',c))
disp('The solution by jacobi matrix method is')
disp(Ujcm)
disp('----------------------------------------')

%-------------------Gauss Seidel method-------------------
-------

Ugs=zeros(n+1,n+1);
%c=0;
R=4+((12.5*(pi^2)*(h^2)));
%while c<=17

    %c=c+1;

    k_max = (n)^3;

tolerance = 1.e-04;
for k = 1:k_max

    frac_diff = 0;
for i=2:1:a+1
    for j=2:1:a+1
        Ugs(i,j) = (((25*(pi^2)*sin(2.5*pi*xy(j-1,1))*sin(2.5*pi*xy(i-1,1)))*(h^2))+Ugs(i-1,j)+Ugs(i,j-1)+Ugs(i,j+1)+Ugs(i+1,j))/R;
        frac_diff = frac_diff + abs(Ugs(i,j)-Uex(i,j));
    end
end
change(k) = frac_diff/(n-2)^2;
if rem(k,100) < 1
    fprintf('Fractional difference is %g after %g steps
', change(k), k);
end
if change(k) < tolerance
    break;
end
end
Ugs
mesh(Ugs);
pause;
semilogy(change);
xlabel('Iteration Number');ylabel('Fractional Changes');

%err_iter_gs=norm(Ugs(:)-Uex(:),inf);

%end

%disp(sprintf('norm_inf error for Gauss Seidel method is =%.5f',err_iter_gs))

%disp(sprintf('number of iteration for Gauss Seidel method is =%.0f',c))

%disp(Ugs)

%disp('---------------------------------------------Gauss Seidel method by LUD matrix-----

c=0;

Ugsm=zeros(n+1,n+1);

Xoldgs=zeros(a*a,1);

Xgsm=zeros(a*a,1);

while c~=17

    c=c+1;

    Xgsm=(inv(L+D)*b)-(inv(L+D))*UU*Xoldgs;

    Xoldgs=Xgsm;
end

r=0;
for i=2:1:n
  for j=2:1:n
    Ugsm(i,j)=Xgsm(r+(j-1),1);
  end
  r=r+n-1;
end

err_iter_gsm=norm(Ugsm(:)-Uex(:),inf);
disp(sprintf('norm_inf error for Gauss Seidel method is =%.5f',err_iter_gsm))
disp(sprintf('number of iteration for Gauss Seidel method is =%.0f',c))
disp('The solution by Gauss Seidel matrix method is')
disp(Ugsm)
disp('-----------------------------------------------')
%-------------------
SOR======================================
Usor=zeros(n+1,n+1);
c=0;
w = \frac{2}{1 + \sin(\pi h)} - .25

R = 4 + ((12.5 \cdot (\pi^2) \cdot (h^2)))

k_{\text{max}} = (a+1)^3

tolerance = 1.e-04;

for k = 1:k_{\text{max}}

\text{frac\_diff} = 0;

for i = 2:a+1

for j = 2:a+1

Usor(i,j) = (w \cdot (\frac{((25 \cdot (\pi^2) \cdot \sin(2.5 \cdot \pi \cdot \text{xy}(j-1,1)) \cdot \sin(2.5 \cdot \pi \cdot \text{xy}(i-1,1)) \cdot (h^2))}{R}) + ((1-w) \cdot Usor(i,j)) + \frac{w \cdot (Usor(i-1,j) + Usor(i+1,j) + Usor(i,j+1) + Usor(i,j-1))}{R};

\text{frac\_diff} = \text{frac\_diff} + \text{abs}(Usor(i,j) - Uex(i,j));

end

end

change(k) = \text{frac\_diff}/(a+1-2)^2;

if \text{rem}(k,100) < 1

fprintf('Fractional difference is %g after %g steps\n', change(k), k);

end
if change(k) < tolerance
    break;
end

Usor

mesh(Usor);

pause;

semilogy(change);

xlabel('Iteration Number'); ylabel('Fractional Changesor');

B.10 Matlab code for irregular region/Chapter Four

format short e;

N1=4;

N2=2;

L1=.4;

L2=.2;

h=L1/N1;

k=L2/N2;

U=zeros(N2+1,N1+1);

A=[-4 1 0 1 0;1 -4 1 0 1;0 2 -4 0 0;2 0 0 -4 1;0 2 0 2 -4];
b=[-8;-4;-8.0565685;-4.02;-0.0565685];

disp('The solution by matrix')

X=inv(A)*b;

disp(X);

disp('------------------------------------------')

for i=1:1:N1+1
    U(3,i)=4;
end

for i=1:1:N2+1
    U(i,1)=4;
end

for i=2:1:N1
    U(2,i)=X(i-1,1);
end

for i=2:1:N1-1
    U(1,i)=X(i+2,1);
end

%====================================================================

==
L=zeros(N1+1,N1+1);
UU=zeros(N1+1,N1+1);
D=zeros(N1+1,N1+1);

for i=1:1:N1+1
    for j=1:1:N1+1
        if i==j
            D(i,j)=A(i,j);
        elseif i<j
            UU(i,j)=A(i,j);
        elseif i>j
            L(i,j)=A(i,j);
        end
    end
end

%========================Jacobi Method==================
Xjc=ones(N1+1,1);
Xnew=zeros(N1+1,1);
c=0;
while c~=100
c=c+1;

Xnew(1,1)=(-b(1,1)+Xjc(2,1)+Xjc(4,1))/4;
Xnew(2,1)=(-b(2,1)+Xjc(1,1)+Xjc(3,1)+Xjc(5,1))/4;
Xnew(3,1)=(-b(3,1)+2*Xjc(2,1))/4;
Xnew(4,1)=(-b(4,1)+2*Xjc(1,1)+Xjc(5,1))/4;
Xnew(5,1)=(-b(5,1)+2*Xjc(2,1)+2*Xjc(4,1))/4;

Xjc=Xnew;

err_iter_jc=norm(Xjc(:)-X(:),inf);

disp(sprintf(' number of iteration for jacobi method is=%.0f',c))

disp(sprintf('norm of error for jacobi method is=%.10f',err_iter_jc))

disp('The solution by jacobi Method is')

disp(Xjc)

disp('---------------------------------------')

c=0;

Xoldjc=zeros(N1+1,1);

Xjcm=zeros(N1+1,1);
while c~=100

c=c+1;

Xjcm=(inv(D)*b)-((inv(D))*((L+UU)*Xoldjc));

Xoldjc=Xjcm;
end

err_iter_jcm=norm(Xjcm(:)-X(:),inf);

disp(sprintf(' number of iteration for jacobi method is=%.0f',c))

disp(sprintf('norm of error for jacobi method is=%.10f',err_iter_jcm))

disp('The solution by jacobi Method is')
disp(Xjcm)
disp('---------------------------------------')

%====================Gauss Seidel======================

Xgs=ones(N1+1,1);

c=0;

while c~=50

c=c+1;

Xgs(1,1)=(-b(1,1)+Xgs(2,1)+Xgs(4,1))/4;
\[
Xgs(2,1) = \frac{-b(2,1) + Xgs(1,1) + Xgs(3,1) + Xgs(5,1)}{4};
\]
\[
Xgs(3,1) = \frac{-b(3,1) + 2Xgs(2,1)}{4};
\]
\[
Xgs(4,1) = \frac{-b(4,1) + 2Xgs(1,1) + Xgs(5,1)}{4};
\]
\[
Xgs(5,1) = \frac{-b(5,1) + 2Xgs(2,1) + 2Xgs(4,1)}{4};
\]

\[
\text{end}
\]

\[
\text{err_iter_gs}=\text{norm}(Xgs(:)-X(:),\text{inf});
\]

\[
\text{disp(sprintf(' number of iteration for Gauss Seidel method is=\%.0f',c))}
\]

\[
\text{disp(sprintf('norm of error for Gauss Seidel method is=\%.10f',err_iter_gs))}
\]

\[
\text{disp('The solution by Gauss Seidel Method is')}
\]

\[
\text{disp(Xgs)}
\]

\[
\text{disp('--------------------------------------')}
\]

\[
c=0;
\]

\[
Xoldgs=\text{zeros}(N1+1,1);
\]

\[
Xgsm=\text{zeros}(N1+1,1);
\]

\[
\text{while } c<50
\]

\[
c=c+1;
\]

\[
Xgsm=(\text{inv}(L+D)*b)-((\text{inv}(L+D))*UU*Xoldgs);
\]
Xoldgs=Xgsm;

end

err_iter_gsm=norm(Xgsm(:)-X(:),inf);

disp(sprintf(' number of iteration for Gauss Seidel method is=%.0f',c))

disp(sprintf('norm of error for Gauss Seidel method is=%.10f',err_iter_gsm))

disp('The solution by Gauss Seidel Method is')

disp(Xgsm)

disp('--------------------------------------')

%%-----------------------SOR method---------------------

Xsor=ones(N1+1,1);

c=0;

w=2/(1+sin(pi*h))

while c~=35
    c=c+1;
    Xsor(1,1)=(1-w)*Xsor(1,1)+(w*(-
b(1,1)+Xsor(2,1)+Xsor(4,1)))/4;
\begin{verbatim}
Xsor(2,1) = (1-w)*Xsor(2,1) + (w*(-b(2,1)+Xsor(1,1)+Xsor(3,1)+Xsor(5,1)))/4;

Xsor(3,1) = (1-w)*Xsor(3,1) + (w*(-b(3,1)+2*Xsor(2,1)))/4;

Xsor(4,1) = (1-w)*Xsor(4,1) + (w*(-b(4,1)+2*Xsor(1,1)+Xsor(5,1)))/4;

Xsor(5,1) = (1-w)*Xsor(5,1) + (w*(-b(5,1)+2*Xsor(2,1)+2*Xsor(4,1)))/4;

end

err_iter_sor = norm(Xsor(:)-X(:),inf);

disp(sprintf(' number of iteration for SOR method is=%.0f',c))

disp(sprintf('norm of error for SOR method is=%.10f',err_iter_sor))

disp('The solution by SOR Method is')

disp(Xsor)

disp('--------------------------------------')

disp('The solution of U(x,y) is=')

disp(U)

B.11 Parent routine for Laplace's Eq. in polar

u=zeros(6,5);
\end{verbatim}
$r = [1 \ 2 \ 4 \ 6 \ 8 \ 1]$;

$rs = r .* r$;

$s = \pi / 8$;

$w = .2$;

$ss = s * s$;

$ws = w * w$;

$s1 = 2 * (1 / ws + 1.0 ./ (ss * rs))$;

$u(6,:) = 100$;

for $k = 1:100$

    $changesum = 0$;

    $ueval = u$;

    for $i = 2:5$

        $T = (u(i-1,1) + u(i+1,1)) / ws + (u(i+1,1) - u(i-1,1)) / (2 * w * r(i))$;

        $u(i,1) = (T + 2 * u(i,2) / (rs(i) * ss)) / s1(i)$;

    end

    for $i = 2:5$

        for $j = 2:4$

The code snippet appears to be a numerical implementation, possibly related to solving a system of equations or a differential equation, given the presence of loops and arithmetic operations. The specific context is not clear from the code alone.
\[ T = \frac{(u(i-1,j) + u(i+1,j))}{ws} + \frac{(u(i+1,j) - u(i-1,j))}{2w*r(i)} \] 

\[ u(i,j) = \frac{(T + u(i,j-1) + u(i,j+1))}{(rs(i)*ss)/s1(i)} \] 

\[ \text{changesum} = \text{changesum} + \text{abs}(ueval-u) \] 

end 

end 

if max(abs(ueval-u)) < .001, break, end 

end 

fprintf('# of iteration =%g
',k) 

disp('u ='); disp(u) 

end 

mesh(u); 

pause; 

semilogy 

xlabel('Iteration Number '); ylabel('Error');
References


بسم الله الرحمن الرحيم

لا يوجد نص يمكن قراءته بشكل طبيعي من الصورة المقدمة.
تمت دراسة المعادلات التفاضلية الجزئية البيضاوية من الدرجة الثانية باستخدام بعض الطرق العددية. لهذا النوع من المعادلات التفاضلية الجزئية تطبيقات في ظواهر الطبيعية والهندسية. في معظم التطبيقات نستخدم عادة قواعد من الرتبة الأولى والثانية لكننا في هذه الرسالة استخدمنا قواعد من رتب أعلى مثل: قاعدة النقط السبعة وقاعدة النقط التسعة. باستخدام هذه القواعد يمكن تحويل المعادلة التفاضلية الجزئية إلى معادلة الفروق الدقيقة، ولحل مثل هذه المعادلة نستخدم طرق التكرار مثل:

(SOR, Jacobi, Gauss Seidel, Multigrid Methods)

في هذا البحث وجدنا أن طريقة (Multigrid Methods) هي الطريقة المثلى من بين جميع الطرق الأخرى، فالوقت المستغرق في هذه الطريقة هو من الرتبة الثالثة بينما هو من الرتبة الخامسة في الطرق الأخرى.