Mathematical Modeling of some Differential Equations Using Numerical Methods

Submitted to Department of Mathematics and Computer Science, Faculty of Science, Menoufia University in Partial Fulfillment of the requirements for the Degree of B. Sc. in Mathematics

***By***

## Students in the Department of Mathematics and Computer Science, Faculty of Science, Menoufia University

***Supervisor***

## Dr. Adel Mohamed Morad

Lecturer of Applied Mathematics, Department of Mathematics and Computer Science, Faculty of Science, Menoufia University

|  |  |
| --- | --- |
| **Contents** |  |
| **Acknowledgments…………….……………………………………** | 4 |
| **Preface………………………………………………………………** | 5 |
| **Chapter One:** Finite Difference Method for Solving Elliptic Partial  Differential Equations**………………………………………** | 6 |
| 1.1 Discretization of Elliptic PDE by Finite Difference Method........ | 7 |
| 1.2 The Principle of Finite Difference Method…………………… | 8 |
| 1.2.1 Taylor's Theorem……………………………………………. | 8 |
| 1.3 Strategy of Discretization…………………………………….. | 9 |
| 1.4 The finite difference algorithm……………………………… | 9 |
| 1.5 Elliptic PDE subject to Boundary Conditions………………… | 13 |
| 1.5.1 Laplace equation with Dirichlet Boundary Conditions……… | 14 |
| 1.5.2 Poisson equation with Dirichlet boundary condition………… | 18 |
| 1.5.3 Laplace equation with Neumann Boundary Conditions…….. | 19 |
| 1.5.4 Poisson equation with Neumann Boundary Conditions……. | 21 |
| **Chapter two:** Applications of Some Physical Problems |  |
| 2.1Laplace Equation by Jacobi Method…………………………… | 22 |
| 2.2 Poisson Equation by Jacobi Method……………………………. | 26 |
| **Chapter three :**KdV equation…………………………………. | 32 |
| 3.1Introduction of KdV……………………………………………. | 33 |
| 3.2 Historical Background………………………………………… | 34 |
| 3.3 The solution of KdV equation including……………………… | 35 |
| 3.3.1 Numerical solution…………………………………….... | 35 |
| 3.3.2 Exact solution…………………………………………. | 38 |
| 3.4 KdV Burger equation……………………………………. | 42 |
| 3.4.1 Mathematical Formulation………………………………… | 42 |

|  |  |
| --- | --- |
| 3.4.2 Classical explicit finite difference method (CEFD)……… | 43 |
| 3.4.3 Analytical solution………………………………………… | 44 |
| 3.4.4 Numerical results and discussion……………………….. | 46 |
| 3.5 Application of KDV……………………………………… | 46 |
| 3.6 Comparison between Experiential, CEFD and exact solution of  KdV .B equation at t=0.002………………………………… | 48 |
| 3.7 Comparison between Experiential, CEFD and exact solution of  KdV .B equation at t=0.001………………………………… | 48 |
| **Chapter Four:** Computational programs for solving PDE |  |
| 4.1 MATLAB……………………………………………………… | 54 |
| 4.1.1 Useful function and operations in MATLAB……………… | 54 |
| 4.1.2 Obtaining help on MATLAB commands…………………… | 55 |
| 4.1.3 Variables in MATLAB……………………………………… | 56 |
| 4.1.4 How to plot with MATLAB……………………………….. | 57 |
| 4.2 Computational Programs for Solving Differential Equations  by MATLAB…………………………………………… | 60 |
| 4.3An introduction to Mathematica……………………………. | 74 |
| 4.3.1 A brief overview of Mathematica…………………… | 74 |
| 4.3.2Symbolic computation………………………………….......... | 75 |
| 4.3.3Graphical…………………………………………. | 75 |

# Acknowledgments

First of all we would like to thank ALLAH for helping us to complete this work. This work would not have been possible if it weren’t for the help, guidance, and friendship of many people.

Deep thanks to Dr. Adel Mohamed Morad and Dr. Ehab Said Selima Department of Mathematics and Computer Science, Faculty of Science, Menoufia University, have been an invaluable source of guidance, patience, and support during elaborating this work. Their continuous assistance and valuable discussion will never be forgotten.

Thanks for Prof. Dr. Mohamed Amin (the head of Department of Mathematics and Computer Science), who put their facilities under our disposal which will never be forgotten.

We are highly appreciative of the support and encouragement provided by our colleagues and our friends in the Department of Mathematics and Computer Science.

Finally, we would like to thank our parents and all our family members for their support and caring attitude which kept us going until the end our work.

# Preface

Elliptic partial differential equations appear frequently in various fields of science and engineering. These involve equilibrium problem and steady state phenomena. The most common examples of equations are the poisons, Laplace and KdV equations. These equations are classified as second and third order liner partial differential equations. Most of these physical problems are very hard to solve analytically, instead, they can be solved numerically using computational methods. In this work, boundary value problems involving Poisson's, Laplace and KdV equations with different type of boundary conditions are solved numerically using the finite difference method (FDM) by MATLAB software and Mathematica.

.

# Chapter One

**Finite Difference Method for Solving Elliptic Partial Differential Equations**

Problems of physics and engineering elliptic partial differential equations can be used to describe such equilibrium difficulties and steady state events (independent of time) (elliptic PDEs). These equations describe how such issues behave. The most common use of second order linear partial differential equations is in

𝜕2𝑢

𝐴 𝜕𝑥

+ 𝐵

𝜕2

𝜕𝑥𝜕𝑦

+ 𝐶

𝜕2𝑢

𝜕𝑦

+ 𝐷

𝜕𝑢

𝜕𝑥

+ 𝐸

𝜕𝑢

𝜕𝑦

+ 𝐹𝑢 = 𝐺(𝑥, 𝑦)

or simply

Auxx + Buxy + Cuyy + Dux + Euy + Fu = 𝐺(𝑥, 𝑦) (1.1)

where A, B, C, D, E, F and the free term G are the coefficients of Eq.(1.1), which can be constants or functions of two independent variables x and y, and u is the unknown function of two independent variables x and y. Equation (1.1) is classified into three types depending on the discriminate

(𝐵2 − 4𝐴𝐶)as follows :

1. Hyperbolic if the discriminate is positive(𝐵2 − 4𝐴𝐶) > 0
2. Parabolic if the discriminate is zero(𝐵2 − 4𝐴𝐶) = 0
3. Elliptic if the discriminate is negative (𝐵2 − 4𝐴𝐶) < 0

We will deal with elliptic PDEs (or in general, with steady state problems) with respect to two types of boundary conditions. These conditions are:

# Dirichlet Boundary Condition:

# The condition under which the unknown function's value is dictated on the domain's boundary.

# Neumann Boundary Condition:

The condition where the value of the normal derivative ( 𝜕𝑢) is

𝜕𝑛

given on the boundary of the domain.

In this project, we solve elliptic partial differential equations in two dimensions using Finite Difference and Finite Element methods, such as the Laplace equation and the Poisson equation. When these techniques are applied to solve elliptic PDEs, they yield a system of linear equations that must be solved iteratively using Jacobi, Gauss-Seidel, successive over Relaxation (SOR), and Conjugate Gradient methods.

# Discretization of Elliptic PDE by Finite Difference Method

The Finite Difference Approach (FDM) is a well-known method for approximating partial differential equations solutions. It was already known in one dimension of space by L. Euler (1707-1783) and was most likely extended to two dimensions by C. Range (1856-1927). When the domain of the problem has regular-shaped borders, this strategy works well. We will use the finite difference approach with a rectangular domain and regular boundary shapes in this project.

# The Principle of Finite Difference Method:

The idea of FDM is to replace the partial derivatives of dependent variable (unknown function) with partial differential equation using finite difference approximations with 𝑂(ℎ𝑛) errors. This procedure converts the region (where the independent variables in PDE are defined on) to a mesh grid of points where the dependent variables derivatives with difference approximation formulas depends on Taylor's Theorem. So, Taylor's Theorem is introduced.

## Taylor's Theorem

Let 𝑢(𝑥) has n continuous derivatives over the interval (𝑎, 𝑏). Then, for a < 𝑥0,𝑥0 + ℎ < 𝑏, we can write the value of u(x) and its derivatives nearby the point 𝑥0+has follows

u(𝑥

+ ℎ) = u(𝑥

) + h 𝑢𝑥(𝑥0) + ℎ2 𝑢𝑥𝑥(𝑥0) + ℎ3 𝑢𝑥𝑥𝑥(𝑥0) + ⋯ +

0 0 1! 2! 3!

ℎ𝑛−1 𝑢𝑛−1(𝑥0) + 𝑂(ℎ𝑛), (1.2)

(𝑛−1)!

where

1. (𝑢𝑥(𝑥0)) is the first derivative with respect to x at the point.
2. (𝑢𝑛−1(𝑥0)) is the n-1th derivative with respect to x at the point.
3. 𝑂(ℎ𝑛) [It is an order h to the n] is an unknown error term that satisfies the property for

𝑓(ℎ) = 𝑂(ℎ𝑛)

𝑓(ℎ)

For any non-zero constant C

lim

ℎ

ℎ𝑛 = 𝐶

When we eliminate the error term𝑂(ℎ𝑛), from the right-hand side of Eq. (1.2), we get an approximation to 𝑢(𝑥0 + ℎ).

# Strategy of Discretization

Using finite difference method to discredited elliptic PDE with its boundary conditions, we can consider the following Poisson equation:

∇2𝑢(𝑥, 𝑦)=𝜕2𝑢 (𝑥, 𝑦) + 𝜕2𝑢 (𝑥, 𝑦) = 𝐺(𝑥, 𝑦)

𝜕𝑥2 𝜕𝑦2

Or we can simply write this equation in another form as:

𝑢𝑥𝑥 + 𝑢𝑦𝑦 = 𝐺(𝑥, 𝑦), 𝑓𝑜𝑟 (𝑥, 𝑦) ∈ ℝ (1.3) The rectangular domain R = {(x, y) | a < 𝑥 < 𝑏, 𝑐 < 𝑦 < 𝑑} and

𝑢(𝑥, 𝑦) = 𝑔(𝑥, 𝑦) for any (𝑥, 𝑦)ϵ 𝑆, where: 𝑆 denotes the boundary of a region 𝑅 ϵ 𝐺(𝑥, 𝑦) is a continuous function on R and 𝑔(𝑥, 𝑦) is continuous on S The continuity of both G and g generated a unique solution of Eq. (1.3). Now, we will use the finite difference algorithm for solving elliptic PDE, like Eq. (1.3).

* 1. **The Finite Difference Algorithm Step 1:** Choose positive integers n and m.

**Step 2:** Define ℎ = 𝑏−𝑎 and𝑘 = 𝑑−𝑐

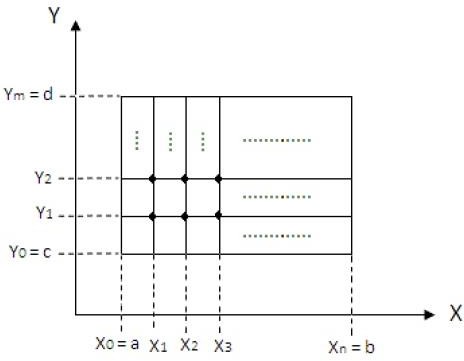
𝑛 𝑚

This step partitions the interval [𝑎, 𝑏] into 𝑛 equal parts of width hand partitions the interval [𝑐, 𝑑] into equal parts of width k as step 3 illustrates.

𝑥𝑖 = 𝑎 + 𝑖 ℎ, 𝑖 = 0,1,2, , 𝑛

𝑦𝑗 = 𝑐 + 𝑗 𝑘, 𝑗 = 0,1,2, . . . . . , 𝑚

Step 2 and step 3 are illustrated in figure 1.1.



# Figure 1.1

It is clear from figure 1.1 that we have horizontal and vertical lines inside the rectangle R these lines are called "grid lines" and their intersections are called "mesh points" of the grid. For each mesh point inside the grid, (𝑥𝑖, 𝑦𝑗),

Where 𝑖 = 1, 2, . . . , 𝑛 − 1 and 𝑗 = 1, 2, . . . , 𝑚 − 1 [2], [21]. We use Taylor series in the variable x about𝑥𝑖to generate the central-difference formula:

𝑢 (𝑥 , 𝑦 ) = 𝑢(𝑥𝑖+1,𝑦𝑗)−2𝑢(𝑥𝑖,𝑦𝑗)+𝑢(𝑥𝑖−1,𝑦𝑗) − ℎ2 . 𝜕4𝑢 {£ , 𝑦 } (1.4)

𝑥𝑥

𝑖 𝑗

ℎ2

Where𝜉𝑖 = (𝑥𝑖−1, 𝑥𝑖+1)

12 𝜕𝑥4

𝑖 𝑗

Also, we use Taylor series in the variable 𝑦𝑗 a bout to generate the central-difference formula:

𝑢 (𝑥 , 𝑦 ) = 𝑢(𝑥𝑖,𝑦𝑗+1)−2𝑢(𝑥𝑖,𝑦𝑗)+𝑢(𝑥𝑖,𝑦𝑗−1) − 𝑘2 . 𝜕4𝑢 (𝑥 , 𝜂

) (1.5)

𝑦𝑦

𝑖 𝑗

𝑘2

12 𝜕𝑦4

𝑖 𝑗

Where 𝜂𝑗𝜖(𝑦𝑖−1, 𝑦𝑖+1) [15].

By inserting Eq. (1.4) and Eq. (1.5) into Eq. (1.3), We get:

𝑢(𝑥𝑖+1,𝑦𝑗)−2𝑢(𝑥𝑖,𝑦𝑗)+𝑢(𝑥𝑖−1,𝑦𝑗) − ℎ2 ∗ 𝜕4𝑢 (𝜉 , 𝑦 ) +

ℎ2 12

𝜕𝑥4

𝑖 𝑗

𝑢(𝑥𝑖,𝑦𝑗+1)−2𝑢(𝑥𝑖,𝑦𝑗)+𝑢(𝑥𝑖,𝑦𝑗−1) − 𝑘2 ∗ 𝜕4𝑢 (𝑥 , 𝜂

) = 𝐺(𝑥 , 𝑦 ) (1.6)

𝑘2

12 𝜕𝑦4

𝑖 𝑗

𝑖 𝑗

for each 𝑖 = 1, 2, 3, . . . , 𝑛 − 1 and 𝑗 = 1, 2, 3, . . . , 𝑚 − 1

# The boundary conditions are:

1- 𝑢(𝑥0, 𝑦𝑗) = 𝑢(𝑥𝑖, 𝑦𝑗)𝑗 = 0,1,2,3, … … , 𝑚

2- 𝑢(𝑥𝑛, 𝑦𝑗) = 𝑢(𝑥𝑛, 𝑦𝑗)𝑗 = 0,1,2,3, … … , 𝑚

3- 𝑢(𝑥𝑖, 𝑦0) = 𝑢(𝑥𝑖, 𝑦0) 𝑖 = 1,2,3, … … , 𝑛 − 1

4- 𝑢(𝑥𝑖, 𝑦𝑚) = 𝑢(𝑥𝑖, 𝑦𝑚)𝑖 = 1,2,3 … … , 𝑛 − 1

Now, by rearranging Eq. (1.6), we get:

−2𝑢(𝑥𝑖, 𝑦𝑗)

ℎ2 +

−2𝑢(𝑥𝑖, 𝑦𝑗)

𝑘2 +

𝑢(𝑥𝑖+1, 𝑦𝑗) + 𝑢(𝑥𝑖−1, 𝑦𝑗) ℎ2

𝑢(𝑥𝑖, 𝑦𝑗+1) + 𝑢(𝑥𝑖, 𝑦𝑗−1)

+ 𝑘2

ℎ2

= 12 ∗

𝜕4𝑢

𝜕𝑥4 (𝜉𝑖, 𝑦𝑗) +

𝑘2

12 ∗

𝜕4𝑢

𝜕𝑦4 (𝑥𝑖, 𝜂𝑗) + 𝐺(𝑥𝑖, 𝑦𝑗)

Or it can simply be written as

−1 1

2 [ 𝑘2 − ℎ2] 𝑢(𝑥𝑖, 𝑦𝑗) +

𝑢(𝑥𝑖+1, 𝑦𝑗) + 𝑢(𝑥𝑖−1, 𝑦𝑗) ℎ2

𝑢(𝑥𝑖, 𝑦𝑗+1) + 𝑢(𝑥𝑖, 𝑦𝑗−1)

+ 𝑘2

ℎ2 𝜕4𝑢 𝑘2 𝜕4𝑢

= 12 𝜕𝑥4 (£𝑖, 𝑦𝑗) + 12 𝜕𝑦4 (𝑥𝑖, µ𝑗) + 𝐺(𝑥𝑖, 𝑦𝑗)

Multiplying both sides by -h², we get:

ℎ 2

2 [(𝑘)

+ 1] 𝑢(𝑥𝑖, 𝑦𝑗) − [(𝑢(𝑥𝑖+1, 𝑦𝑗) + 𝑢(𝑥𝑖−1, 𝑦𝑗)]

ℎ 2

− (𝑘)

[𝑢(𝑥𝑖, 𝑦𝑗+1) + 𝑢(𝑥𝑖, 𝑦𝑗−1)]

ℎ2

= −ℎ2 [

𝜕4

(𝜉𝑖, 𝑦𝑗) +

𝑘2 𝜕4𝑢

(𝑥𝑖, 𝜂𝑗)] − ℎ2𝐺(𝑥𝑖, 𝑦𝑗)

12 𝜕𝑥4 12 𝜕𝑦4

In difference-equation form, this result in the central–difference method with local truncation error 𝑂(ℎ² + 𝑘²).

Simplifying the last equation and letting 𝑢𝑖,𝑗 approximate𝑢(𝑥𝑖, 𝑦𝑗), we get:

ℎ 2 ℎ 2

2 [(𝑘)

+ 1] 𝑢𝑖,𝑗 − [𝑢𝑖+1,𝑗 + 𝑤𝑖−1,𝑗] − (𝑘)

[𝑢𝑖,𝑗+1 + 𝑢𝑖,𝑗−1]

= −ℎ2𝐺(𝑥𝑖, 𝑦𝑗) (1.8)

for each 𝑖 = 1, 2, . . . , 𝑛 − 1 and 𝑗 = 1, 2, . . . , 𝑚 − 1. with boundary conditions:

i. 𝑢0,𝑗 = 𝑔(𝑥0, 𝑦𝑗), ∀𝑗 = 0,1,2, …,m

ii. 𝑢𝑛,𝑗 = 𝑔(𝑥𝑛, 𝑦𝑗), ∀𝑗 = 0,1,2, …,m

iii. 𝑢𝑖,0 = 𝑔(𝑥𝑖, 𝑦0), ∀𝑖 = 1,2 … , 𝑛 − 1 (1.9)

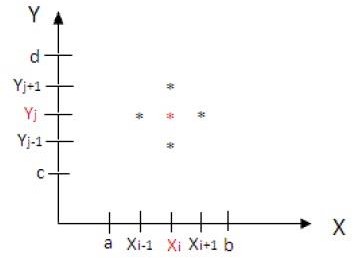
iv. 𝑢𝑖,𝑚 = 𝑔(𝑥𝑖, 𝑦𝑚), ∀𝑖 = 1,2 … , 𝑛 − 1

where𝑢𝑖,𝑗 approximates𝑢(𝑥𝑖, 𝑦𝑗). For more details, see references [2], [3] and [8]. Eq. (1.8) involves approximations to the unknown function

𝑢(𝑥, 𝑦)at the points

(𝑥𝑖, 𝑦𝑗), (𝑥𝑖+1, 𝑦𝑗), (𝑥𝑖−1, 𝑦𝑗), (𝑥𝑖, 𝑦𝑗+1), and (𝑥𝑖, 𝑦𝑗−1)

These points form a star–shape region in the grid (as figure 1.2 shows) which shows that any equation involves approximations about (𝑥𝑖, 𝑦𝑗)



# Figure 1.2

When we use formula (1.8) with boundary conditions (1.9), then at all points (𝑥𝑖, 𝑦𝑗) that is adjacent to (𝑛 − 1) ∗ (𝑚 − 1) 𝑏𝑦 (𝑛 − 1) ∗ (𝑚 − 1) linear system with the unknowns being the approximations

𝑤𝑖,𝑗 𝑡𝑜 𝑢(𝑥𝑖, 𝑦𝑗) at the interior meth Points.

The generated linear system should be solved by Jacobi, Gauss-Seidel, and Successive over Relaxation (SOR), or Conjugate Gradient methods. This system (that involves the unknowns) produces satisfactory results if a relabeling of the interior mesh points is introduced. A favorable labeling of these points is [3], [8] and [20]:

𝑙𝑟 = (𝑥𝑖, 𝑦𝑗)𝑎𝑛𝑑 𝑢𝑟 = 𝑢𝑖,𝑗 𝑤ℎ𝑒𝑟𝑒 𝑟 = 𝑖 + (𝑚 − 1 − 𝑗)(𝑛 − 1),

∀𝑖 = 1,2, … , 𝑛 − 1 𝑎𝑛𝑑 ∀ 𝑗 = 1,2, … , 𝑚 − 1

# Elliptic PDE subject to Boundary Conditions:

The unknown function (dependent variable) must meet certain requirements on the boundary S in order to solve the Laplace equation and the Poisson equation on the boundary of a domain R. With respect to two variables, we'll look at the Laplace equation and the Poisson equation.

border conditions of several kinds Dirichlet and Neumann boundary conditions are what they're called.

# Laplace equation with Dirichlet Boundary Conditions:

When the function is defined on any part of a domain ℝ , then we call this part Dirichlet boundary 𝑠𝐷, i.e. the unknown function 𝑢 is prescribed on the boundary, that is, 𝑢(𝑥, 𝑦) = 𝑔(𝑥, 𝑦), (𝑥, 𝑦) ∈ 𝑆 where the function g is a known function.

𝜕2𝑢 𝜕2𝑢

𝑆𝐷 ∶ 𝑢 = 𝑔

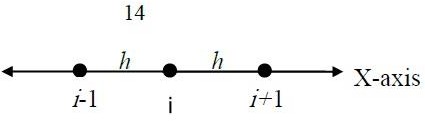
𝜕𝑥2 + 𝜕𝑦2 = 0

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

𝜕2𝑢 + 𝜕2𝑢 = 0 (1.10)

𝜕𝑥2 𝜕𝑦2

We consider three points 𝑖 + 1, 𝑖 and 𝑖 − 1 which are located on X-axis with equal distance h between them (as figure 1.3 shows).



# Figure 1.3

Let the value of the function 𝑢(𝑥, 𝑦) at the points

(𝑖 − 1, 𝑗), (𝑖, 𝑗)And (𝑖 + 1, 𝑗) be 𝑢𝑖−1,𝑗 , 𝑢𝑖,𝑗 , 𝑎𝑛𝑑 𝑢𝑖+1,𝑗, respectively. Now, use Taylor series to express𝑢𝑖+1,𝑗 and 𝑢𝑖−1,𝑗 in the form of Taylor expansions about the point i as follows:

ℎ 𝜕𝑢 ℎ2 𝜕2𝑢 ℎ3 𝜕3𝑢 ℎ4 𝜕4𝑢

𝑢𝑖+1,𝑗 = 𝑢𝑖,𝑗 + 1! 𝜕𝑥 |𝑖 + 2! 𝜕𝑥2 |𝑖 + 3! 𝜕𝑥3 |𝑖 + 4! 𝜕𝑥4 |𝑖

+ 𝑂(ℎ5) (1.11)

h 𝜕𝑢 ℎ2 𝜕2𝑢 ℎ3 𝜕3𝑢 ℎ4 𝜕4𝑢

𝑢𝑖−1,𝑗 = ui,j − 1! 𝜕𝑥 |𝑖 + 2! 𝜕𝑥2 |𝑖 − 3! 𝜕𝑥3 |𝑖 + 4! 𝜕𝑥4 |𝑖

+ 𝑂(ℎ5) (1.12)

By adding Eq. (1.11) and Eq. (1.12), we get:

𝑢𝑖+1,𝑗 − 𝑢𝑖−1,𝑗 = 2ui,j +

ℎ2 𝜕2𝑢 2! 𝜕𝑥2

|𝑖 +

ℎ4 𝜕4𝑢 4! 𝜕𝑥4

|𝑖 + 𝑂(ℎ5)

𝜕2𝑢

By rearranging the above equation, we get:

𝑢𝑖+1,𝑗 − 2𝑢𝑖,𝑗 + 𝑢𝑖−1,𝑗

𝜕𝑥2 |𝑖 =

ℎ2

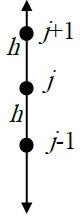
+ 𝑂(ℎ5) (1.13)

Eq. (1.13) is a finite difference approximation formula with error

term 𝑂(ℎ2) of second order for𝜕2𝑢 | .

𝜕𝑥2 𝑖

Now, Similarly, consider three points 𝑗 + 1 , 𝑗 and 𝑗 − 1 which are located on the Y-axis with equal distance ℎ between them (as figure 1.4 shows).



# Y-axis Figure 1.4

Let the value of the function 𝑢(𝑥, 𝑦) at the points(𝑖, 𝑗 + 1), (𝑖, 𝑗), and (𝑖, 𝑗 − 1) be 𝑢𝑖,𝑗−1 , 𝑢𝑖,𝑗 and𝑢𝑖,𝑗+1 respectively. Using Taylor series to express 𝑢𝑖,𝑗+1 𝑎𝑛𝑑 𝑢𝑖,𝑗−1 in the form of Taylor expansions at the point

𝑗,the finite difference approximation formulas with error term of second

order for𝜕2𝑢 | 𝑎𝑛𝑑 𝜕𝑥 |

are, respectively:

𝜕𝑦2 𝑖 𝜕𝑦 𝑗

𝜕𝑥

𝜕𝑦 |𝑗 =

𝑢𝑖,𝑗+1 − 𝑢𝑖,𝑗−1

2ℎ (1.14)

Eq. (1.14) is a finite different approximation formula with error term

𝑂(ℎ2) of second order for 𝜕𝑢 |

𝜕𝑦 𝑖

𝜕2𝑢

𝜕𝑦2

|𝑖 =

𝑢𝑖,𝑗+1 − 2𝑢𝑖,𝑗 + 𝑢𝑖,𝑗−1 ℎ2

+ 𝑂(ℎ2) (1.15)

Now, by combining figure 1.3 and figure 1.4 together, we get the star– shape (or 5-points stencil) region about the point (𝑖, 𝑗) as shown in figure (1.5 )

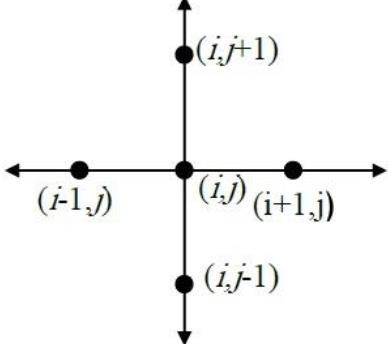


Figure 1.5

Inserting Eq. (1.13) and Eq. (1.15) into Eq. (1.10)

𝜕2𝑢 𝜕2𝑢

𝑢𝑖+1,𝑗 − 2𝑢𝑖,𝑗 + 𝑢𝑖−1,𝑗 𝑢𝑖,𝑗+1 − 2𝑢𝑖,𝑗 + 𝑢𝑖,𝑗−1

(𝜕𝑥2 + 𝜕𝑦2) |𝑖,𝑗 =

= 0

ℎ2 + ℎ2

By rearranging the above equation, we get [12]: (𝑢𝑖+1,𝑗 + 𝑢𝑖,𝑗+1) − 4𝑢𝑖,𝑗 + (𝑢𝑖−1,𝑗 + 𝑢𝑖,𝑗−1) = 0 SO,

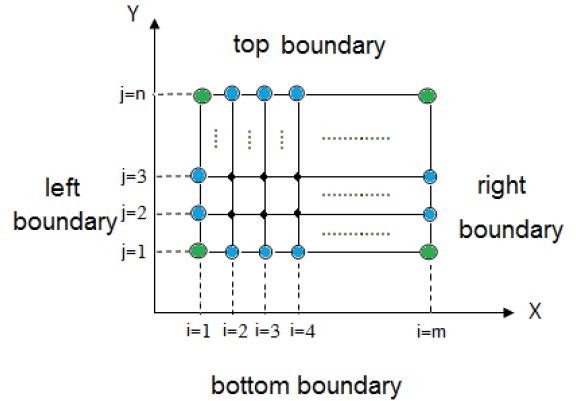
𝑢𝑖,𝑗 =

1

4 (𝑢𝑖+1,𝑗 + 𝑢𝑖,𝑗+1 + 𝑢𝑖−1,𝑗 + 𝑢𝑖,𝑗−1) (1.17)

In general, if 𝑢 satisfies Laplace equation, then u, at any point in the domain R, is the average of the value of 𝑢 at the four surrounding points in the 5-point stencil as shown in figure 1.2.

Now, suppose we have Dirichlet boundary conditions defined on the rectangular domain such that 1 ≤ 𝑖 ≤ 𝑚 𝑎𝑛𝑑 1 ≤ 𝑗 ≤ 𝑛 shown in figure 1.6.



# Figure 1.6

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

𝑢 = 𝑔 is defined on the left, top, right, and bottom boundary walls so that the boundary grid points (blue points) and the corner grid points (green points) are known [3]. In other words, the values of the points

(𝑥𝑖, 𝑦𝑗), ∀𝑖 = 1,2,3, … … 𝑚 𝑎𝑛𝑑, ∀𝑗 = 1,2,3, … … 𝑛 Under the function 𝑔 are known. For the corner grid points, we use the following equations:

𝑢(1,1) =

1

1

[𝑢(2,1) + 𝑢(1,2)]

2

𝑢(𝑚, 1) =

[𝑢(𝑚 − 1,1) + 𝑢(𝑚, 2)]

2

𝑢(

(

1, 𝑛) =

) 1

1 [𝑢

2

(

(1, 𝑛 − 1)

)

+ 𝑢

(2, 𝑛)]

[4, 17]

(1.18)

𝑢 𝑚. 𝑛

= [𝑢

2

𝑚, 𝑛 − 1

+ 𝑢(𝑚 − 1, 𝑛)]

# Poisson equation with Dirichlet boundary condition:

To derive the formula of finite difference approximation with Dirichlet boundary condition for Poisson equation:

𝜕2𝑢 𝜕2𝑢

𝜕𝑥2 + 𝜕𝑦2 = 𝐺(𝑥, 𝑦) (1.19)

Following similar approach for Laplace equation with some amendments in Eq. (1.17), that is [12]:

𝑢𝑖,𝑗 =

1

4 (𝑢𝑖+1,𝑗 + 𝑢𝑖,𝑗+1 + 𝑢𝑖−1,𝑗 + 𝑢𝑖,𝑗−1) −

h2

4 Gi,j (1.20)

# Laplace Equation with Neumann Boundary Conditions:

When the normal derivative of the unknown function 𝑈 is prescribed on the boundary of a domain R, then we call this part Neumann boundary

𝑆𝑁

𝑖. 𝑒 the value of the normal derivative 𝜕𝑢 = 𝑔(𝑥, 𝑦) is given on the

𝜕𝑛

boundary of the domain, where 𝑔(𝑥, 𝑦) is a given function.

𝜕2𝑢

𝜕𝑥2 + 𝜕𝑦2 = 𝐺

𝜕2𝑢

𝜕𝑢

𝑆𝑁: 𝜕𝑛 = 𝑔(𝑥, 𝑦)

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

𝜕2𝑢 𝜕2𝑢

𝜕𝑥2 + 𝜕𝑦2 = 0

Consider that we have a rectangle domain as shown in figure 1.7

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

𝜕𝑢

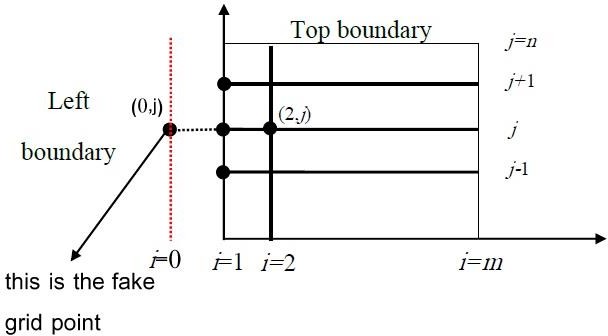
𝜕𝑛

𝜕𝑢

= 𝜕𝑥

= −g(y) (1.21)

Now, we want to approximate Eq. (1.21) using the second order approximation using Eq. (1.14). This procedure puts the grid points (1, 𝑗) outside the domain towards the left that is located on imaginary boundary (red line) that their fake coordinates will be (0, 𝑗)[4] , [8].



# Figure 1.7

So, Eq. (1.21) is approximated using Eq. (1.14) at the line 𝑖 = 1

Thus,

𝜕𝑢

𝜕𝑥 |(1,𝑗) =

𝑢1+1,𝑗 − 𝑢1−1,𝑗

2ℎ =

𝑢(2, 𝑗) − 𝑢(0, 𝑗)

2ℎ = −𝑔(1, 𝑗)

𝑢(0, 𝑗) = 𝑢(2, 𝑗) + 2ℎ𝑔(1, 𝑗)

Now, we write Eq. (1.17) at the point (1, 𝑗) as

𝑢 = 1 (𝑢

+ 𝑢

+ 𝑢

+ 𝑢

) (1.22)

1,𝑗 4 1+1,𝑗

1

1,𝑗+1

1−1,𝑗

1,𝑗−1

𝑢(1, 𝑗) = 4 [𝑢2,𝑗 + 𝑢1,𝑗+1 + 𝑢0,𝑗 + 𝑢1,𝑗−1]

1

= 4 [𝑢(2, 𝑗) + 𝑢(1, 𝑗 + 1) + 𝑢(0, 𝑗) + 𝑢(1, 𝑗 − 1)]

By substituting Eq. (1.22) in the previous formula, we get:

1

𝑢𝑖,𝑗 = 4 [𝑢(2, 𝑗) + 𝑢(1, 𝑗 + 1) + 𝑢(2, 𝑗) + 2ℎ𝑔(1, 𝑗) + 𝑢(1, 𝑗 − 1)]

1

= 𝑢𝑖,𝑗 = 4 [2𝑢(2, 𝑗) + 2ℎ𝑔(1, 𝑗) + 𝑢(1, 𝑗 + 1) + 𝑢(1, 𝑗 − 1)] (1.23)

For any two positive integers m and n, we use Eq. (1.23) for 2 ≤ 𝑗 ≤ 𝑛 − 1, where 𝑔(1, 𝑗) is a specified function. As Dirichlet condition is specified on north, east and south walls, the values.

{𝑢(𝑖, 𝑛), 2 ≤ 𝑖 ≤ 𝑚 − 1}, {𝑢(𝑚, 𝑗), 2 ≤ 𝑗 ≤ 𝑛 − 1}, 𝑎𝑛𝑑 {𝑢(𝑖, 1), 2 ≤

𝑖 ≤ 𝑚 − 1}are known .

To find the values of corner grid points, we use Eq. (1.18).

# Poisson equation with Neumann Boundary Conditions:

Consider the Poisson equation:

𝜕2𝑢 𝜕2𝑢

𝜕𝑥2 + 𝜕𝑦2 = 𝐺(𝑥, 𝑦)

With Neumann boundary condition:

𝜕𝑢 𝜕𝑢

𝜕𝑛 = 𝜕𝑥 = −g(y)

Similar to Laplace equation, the difference approximation formula of Neumann condition at the fake (ghost) grid point (0, 𝑗) is Eq. (1.22), that is:

𝑢(0, 𝑗) = 𝑢(2, 𝑗) + 2ℎ𝑔(1, 𝑗)

Now, using Eq. (1.20) to find the value of the point (1, 𝑗), we get:

1

𝑢1,𝑗 = 4 [𝑢(2, 𝑗) + 𝑢(1, 𝑗 + 1) + 𝑢(0, 𝑗) + 𝑢(1, 𝑗 − 1)] −

ℎ2

4 𝐺1,𝑗 (1.25)

By substitute Eq. (1.22) into Eq. (1.25) with 𝑢(0, 𝑗) = 𝑢0.𝑗 get:

𝑢1,𝑗 =

SO

𝑢1,𝑗 =

1

4 (𝑢2,𝑗 + 𝑢1,𝑗+1 + 𝑢2,𝑗 + 2ℎ𝑔1,𝑗 + 𝑢1,𝑗−1) −

1

4 (𝑢2,𝑗 + 𝑢1,𝑗+1 + 𝑢2,𝑗 + 2ℎ𝑔1,𝑗 + 𝑢1,𝑗−1) −

ℎ2

4 𝐺𝑖,𝑗

ℎ2

4 𝐺1,𝑗

𝑢 = 1 (2𝑢

+ 𝑢

+ 2ℎ𝑔

+ 𝑢

) − ℎ2 𝐺

(1.26)

1,𝑗 4

2,𝑗

1,𝑗+1

1,𝑗

1,𝑗−1

4 1,𝑗

If 𝑖 ≠ 1, we use Eq. (1.20).

Using the same method, we can deal with other boundary points except the corner points. For corner points, we use Eq. (1.18) to find their values.

[4] and [8].

# Chapter two Applications of Some Physical Problems

* 1. **Laplace Equation by Jacobi Method**

Consider the following Laplace equation

𝑢𝑥𝑥 + 𝑢𝑦𝑦 = 0

With square domain R = {(𝑥, 𝑦)|𝑎 = 0 < 𝑥 < 𝑏 = 1,

𝑐 = 0 < 𝑦 < 𝑑 = 1} Subject to Dirichlet boundary conditions given on the boundaries:

𝑢(𝑥, 1) = 𝑥, 𝑢(1, 𝑦) = 𝑦, 𝑎𝑛𝑑 𝑢 (0, 𝑦) = 𝑢(𝑥, 0) = 0

*As shown in figure 2.1*.

We want to approximate the solution u by using Finite Difference Algorithm:

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

**Step 2**: Define ℎ = 𝑏−𝑎 = 1−0 =

1 𝑎𝑛𝑑𝑘 =

𝑑−𝑐 = 1−0 = 1

𝑛 3 3 𝑚 3 3

On X–axis, the interval [0,1] is divided into 𝑛 = 3 equal parts of

width ℎ = 1 also the interval [0,1] is divided, on Y–axis, into 𝑚 = 3

3

equal parts of width

𝑘 = 1.

3

**Step 3**: Define the (horizontal) grid lines as

𝑥𝑖 = 𝑎 + 𝑖ℎ, 𝑖 = 0,1,2, 𝑛 = 3

1

for i = 0 ∶ x0 = 0 + (0) (3) = 0 = a

1 1

𝑓𝑜𝑟 𝑖 = 1 ∶ 𝑥1 = 0 + (1) (3) = 3

1 2

𝑓𝑜𝑟 𝑖 = 2 ∶ 𝑥2 = 0 + (2) (3) = 3

1 3

𝑓𝑜𝑟 𝑖 = 3 ∶ 𝑥3 = 0 + (3) (3) = 3 = 1 = 𝑏

In the same Manner we can find the vertical grid lines

𝑦𝑗 = 𝑐 + 𝑗𝑘,

𝑗 = 0, 1, 2 , 𝑚 = 3

where𝑦 = 0 = 𝑐, 𝑦 = 1 , 𝑦 = 2 𝑎𝑛𝑑 𝑦

= 1 = 𝑏

0 1 3 2 3 3

The grid given in the following figure.

# Figure 3.1

The blue points are known boundary points and the green points are corner points that are easy to be calculated by Eq. (1.18).

However, the black (interior) points are not known which are to be

approximate Now, we use the difference equation (1.17) to approximate the interior (black points) mesh points as follows:

1

𝑢𝑖,𝑗 = 4 (𝑢𝑖+1,𝑗 + 𝑢𝑖,𝑗+1 + 𝑢𝑖−1,𝑗 + 𝑢𝑖,𝑗−1)

For 𝑖 = 1 and 𝑗 = 1

1

𝑢1,1 = 4 (𝑢1+1,1 + 𝑢1,1+1 + 𝑢1−1,1 + 𝑢1,1−1)

𝑢 = 1 (𝑢

+ 𝑢

+ 𝑢

+ 𝑢

) (2.1)

1,1 4 2,1

1,2

0,1

1,0

but both 𝑢0,1and𝑢1,0 are known boundary points whereas 𝑢2,1 and 𝑢1,2 are not known. So the value of are 𝑢0,1(on left boundary) and 𝑢1,0 = 0

(on bottom boundary) So the difference equation (2.1) becomes

1

𝑢1,1 = 4 (𝑢2,,1 + 𝑢1,2 + 0 + 0)

4𝑢1,1 − 𝑢2,1 − 𝑢1,2 = 0 (2.2)

We can label these mesh points as follows:

u1,1 = u3, u2,1 = u4, u1,2 = u1 and u2,2 = u2 Where 𝑙 = 𝑖 + (𝑚 − 1 − 𝑗)(𝑛 − 1) ∀𝑖 = 1,2, 𝑎𝑛𝑑 ∀𝑗 = 1,2 After labeling the interior mesh points, then Eq. (2.2) becomes:

−𝑢1 + 4𝑢3 − 𝑢4 = 0 (2.3)

1

𝑓𝑜𝑟 𝑖 = 2 𝑎𝑛𝑑 𝑗 = 1 ∶ −𝑢2 − 𝑢3 + 4𝑢4 = 3 (2.4)

1

𝑓𝑜𝑟 𝑖 = 1 𝑎𝑛𝑑 𝑗 = 2 ∶ 4𝑢1 − 𝑢2 − 𝑢3 = 3 (2.5)

4

𝑓𝑜𝑟 𝑖 = 2 𝑎𝑛𝑑 𝑗 = 1 ∶ −𝑢1 + 4𝑢2 − 𝑢4 = 3 (2.6)

Rearrange the equations (2.3), (2.4), (2.5), and (2.6) then we get:

1

4𝑢1 − 𝑢2 − 𝑢3 = 3

4

−𝑢1 + 4𝑢2 − 𝑢4 = 3

−𝑢1 + 4𝑢3 − 𝑢4 = 0

1

−𝑢2 − 𝑢3 + 4𝑢4 = 3

This linear system could be written in matrix form as, 𝐴𝑢 = 𝑏 where

4 −1−1 0

𝑢1

1

𝖥31

I4I

𝐴 = [−1 4

0 −1] , 𝑢 = [𝑢2] , 𝑎𝑛𝑑 𝑏 = I3I

−1 0

4 −1

𝑢3 I I

0 −1−1 4

𝑢4

I0I I1I [3]

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

𝑢 = (0.222222, 0.444444, 0.111111, 0.222222)𝑇

# Poisson Equation by Jacobi Method

Consider the following Poisson equation

uxx + uyy = x y

With square domain R = {(x, y) | a = 0 < 𝑥 < 𝑏 = 1, 𝑐 = 0 <

𝑦 < 𝑑 = 1} subject to Neumann boundary conditions

∂u

∂n =

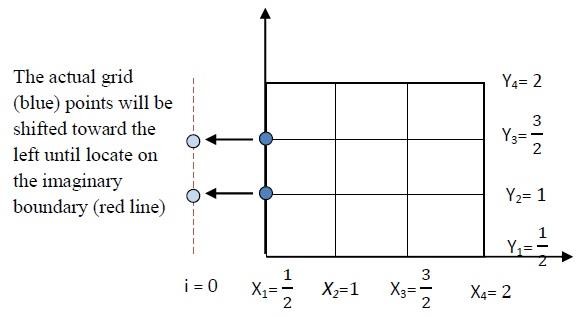
∂u

= g(y) = y

∂x

supplied on the left boundary, and Dirichlet boundary conditions u = 1 on the rest. To estimate the solution of the Poisson equation, we shall use the finite difference approach.The mesh size ℎ = 1as shown in figure 2.2.

2



# Figure 2.2

We want to put the grid points (1, 𝑗) that is the blue points outside the domain toward the left. Let 𝑚 = 𝑛 = 4, the following are known as boundary conditions for

2 ≤ 𝑖 ≤ 4 − 1 𝑎𝑛𝑑 𝑛 = 4

𝑢 (2, 4) = 1, 𝑢 (3, 4) = 1,

𝑎𝑛𝑑 2 ≤ 𝑗 ≤ 4 − 1 𝑎𝑛𝑑 𝑚 = 4

𝑢 (4, 2) = 1, 𝑢 (4, 3) = 1,

𝑎𝑛𝑑 2 ≤ 𝑖 ≤ 4 − 1 𝑎𝑛𝑑 𝑗 = 1

𝑢 (2, 1) = 1, 𝑢 (3, 1) = 1.

Now, we use the following Eq. (1. 26) to approximate the values of boundary points on left boundary:

1

u(1, j) = 4 [2u(2, j) + 2hg1,j + u(1, j + 1) + u(1, j − 1)] −

1

16 Gi,j

for 2 ≤ j ≤ 4 − 1, g1,j = g(x1, yj) = yj and Gi,j = xiyj

1

𝑢(1, 2) = 4 [2𝑢(2, 2) + 2ℎ𝑔1,2 + 𝑢(1, 2 + 1) + 𝑢(1, 2 − 1)] −

1

16 𝑥1𝑦2

1 1 1

𝑢(1, 2) = 4 [2𝑢(2, 2) + 2ℎ𝑔1,2 + 𝑢(1, 3) + 𝑢(1, 1)] − 16 × 2 × 1

But𝑢 (1, 1) is a corner point which we can evaluate its value by the equation:

So,

𝑢(

1, 2) =

1 [2𝑢

4

(2, 2)

+ 𝑢

(1, 3) + 1 +

2

1 𝑢

2

(1, 2)

+ 1] − 1

32

Rearrange this equation, then we get:

7 𝑢(1, 2) − 2𝑢(2, 2) − 𝑢(1, 3) = 11

(2.1)

2 8

𝑛𝑜𝑤, 𝑢(1,3)

1 1

= 4 [2𝑢(2, 3) + 2 × 2 𝑔1,3 + 𝑢(1, 3 + 1) + 𝑢(1, 3 − 1)]

1

− 16 𝑥1𝑦3

𝑢(1, 3) =

1

[2𝑢(2, 3) + 𝑢(1, 4) + 𝑢(1, 2) +

4

3

2] −

1 1 3

16 × 2 × 2

But𝑢 (1, 4) is a corner point which we can evaluate its value by the equation:

So,

𝑢(

1, 3) =

1 [2𝑢

4

(2, 3) + 3

2

+ 1 𝑢

2

(1, 3) + 1

2

+ 𝑢

(1, 2)] − 3

64

Rearrange this equation, then we get:

7 u(1, 3) − 2u(2, 3) − u(1, 2) = 29

(2.2)

2 16

Now, for i = 2, 3 and j = 2, 3, we use eq. (1.20)

𝑢(2, 2) =

1

[2𝑢(3, 2) + 𝑢(1, 2) + 𝑢(2, 3) + 𝑢(2, 1)] −

4

1

16 × 1 × 1

But 𝑢(2, 2) is a known boundary point which is equal to 1

So, substitute its value and then rearrange the equation, then we get

4u(2, 2) − u(3, 2) − u(1, 2) − u(2, 3) = 3(2.3)

4

1 1 3

𝑢(2, 3) =

[𝑢(3, 3) + 𝑢(1, 3) + 𝑢(2, 4) + 𝑢(2, 2)] −

4

16 × 1 × 2

But 𝑢(2, 4) is a known boundary point which is equal to 1

So, substitute its value and then rearrange the equation, then we get

4u(2, 3) − u(3, 3) − u(1, 3) − u(2, 2) = 5(2.4)

8

𝑢(3, 2) =

1 1

[𝑢(4, 2) + 𝑢(2, 2) + 𝑢(3, 3) + 𝑢(3, 1)] −

4 16

3

× 2 × 1

But 𝑢(4, 2) and 𝑢(3, 1) are known boundary points which are equal to 1 So, substitute their values and then rearrange the equation, then we get

4u(3, 2) − u(2, 2) − u(3, 3) = 13(2.5)

8

𝑢(3, 3) =

1 1

[𝑢(4, 3) + 𝑢(2, 3) + 𝑢(3, 4) + 𝑢(3, 2)] −

4 16

3 3

× 2 × 2

But 𝑢(4, 3) and 𝑢(3, 4) are known boundary points which are equal to 1 So, substitute their values and then rearrange the equation, then we get

4u(3, 3) − u(2, 3) − u(3, 2) = 23

16

Now, we have six equations in six variables:

(2.6)

7 11

𝑢(1, 2) − 2𝑢(2, 2) − 𝑢(1, 3) =

2 8

7 29

u(1, 3) − 2u(2, 3) − u(1, 2) =

2 16

3

4u(2, 2) − u(3, 2) − u(1, 2) − u(2, 3) =

4

5

4u(2, 3) − u(3, 3) − u(1, 3) − u(2, 2) =

8

13

4u(3, 2) − u(2, 2) − u(3, 3) =

8

23

4𝑢(3, 3) − 𝑢(2, 3) − 𝑢(3, 2) =

16

Label the variables as follow

u(1, 3) = u1, u(2, 3) = u2, u(3, 3) = u3, u(1, 2) = u4,

u(2, 2) = u5 𝑎𝑛𝑑 u(3, 2) = u6

So, the linear system can be written as:

7 29

2 𝑢1 − 2𝑢2 − 𝑢4 = 16

5

−𝑢1 + 4𝑢2 − 𝑢3 − 𝑢5 = 8

23

−𝑢2 + 4𝑢3 − 𝑢6 = 16

−𝑢1 +

7 11

2 𝑢4 − 2𝑢5 = 8

3

−𝑢2 − 𝑢4 + 4𝑢5 − 𝑢6 = 4

13

−u3 − u5 + 4u6 = 8

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

29

𝖥 1

7

𝖥 2 −2

0 −1 0

0

0 1 u1

𝖥 1

I16I I 5 I

I 8 I

I−1

4 −1 0

−1 0 I u2

I23I

I −1 4

7

0

0 −1I u3

= I16I

I 0 0

I

−

−1 1 0 2

−2 0 I u4

4 −1I Iu5I

I11I

I 8 I

I 0 0

−1−1−1 4

I [u6]

I 3 I

[ 0 0 ] I I

I 4 I I13I [ 8 ]

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

u = (1.4694, 0.9758, 0.8179, 1.3788, 0.9908,0.8584)T

# Chapter three

|  |  |
| --- | --- |
| C:\Users\ascom\Desktop\IMG-20190531-WA0003.jpg  ***Figure 3.1 Dr.D.J.Korteweg*** | C:\Users\ascom\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG-20190531-WA0004.jpg  ***Figure 3.2 Dr.G.DeVries*** |



Figure 3.3: Solitary wave shape of KdV equation in shallow water.

# Introduction

Traveling waves as solutions to the Korteweg-de Vries equation (KdV), a non-linear third-order Partial Differential Equation (PDE), have piqued curiosity for over 150 years. The author's goal is to offer an analytical exact solution to the KdV problem utilising elementary operations and the Backlund transform. Backlunds transform to perform non-linear superposition of many waves, also known as Solitary waves or Solitons, is of particular importance. o. the derivation of the exact answers follows the presentation of Vedenski (1992 ). It's fascinating that these exact solutions, which require a lot of calculations, can be accomplished using the computer algebra mathematical system (see Wolfram 1999), which not only displays analytical expressions but also 3D-Plots, contour plots, and 2D-Plots for discrete time values, animating these 2D-Plots and presenting these plots, as well as the solution formulae, on an internet page. The author's other goal was to use numerical discrete methods to solve non-linear PDEs. The exact solutions obtained here provide an excellent opportunity to assess the quality of numerical approaches by comparing numerical results to the exact answer.

# Historical Background:

A nice tale about the Korteweg-deVries equation's history and underlying physical properties may be found on the Herriot-Watt University's website in Edinburgh (Scotland), see Eilbeck (1998). The following is an excerpt from that page: A young Scottish engineer named John Scott Russell (1808-1882) made a startling scientific discovery while performing trials to identify the most efficient design for canal boats over 150 years ago. I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped-not so the mass of water in the channel which it had set in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smootly shaped elevation.. On horseback, I pursued it and caught up with it as it continued to roll at around eight or nine miles per hour, keeping its original shape of thirty feet long and a foot to a foot and a half in height. Its height steadily decreased, and after a one- or two-mile chase, I lost it in the channel's windings. My first chance encounter with that rare and lovely occurrence, which I have dubbed the wave of translation, occurred in the month of August 1834.

ut(x, t) + 6u(x, t)ux(x, t) + uxxx(x, t) = 0

This is the Kortwege-de Vries equation (KdV) which is nonlinear because of the product shown in the second summand and which is of third order

because of the third derivative as highest in the third summand. The factor 6 is just a scaling factor to make solutions easier to describe.

# The solution of the KdV equation including:

* + 1. **Numerical solution of KdV equation.**

# Exact solution of KdV equation.

* + 1. **Numerical solution of KdV equation: Abstract**

# Implicit difference techniques for nonlinear PDEs, such the Kortwege-de Vries (KdV) equation, necessitate the solution of huge systems of equations at each time step, resulting in significant computation times. A group of finite difference schemes for KdV are provided, and a parallel algorithm for one of them is proposed and implemented. The performance and scalability of the technique are examined after a computational model is compared to experimental results.

# Introduction:

u(x, t) = ut(x, t) + 6u(x, t)ux(x, t) + uxxx(x, t) = 0 f(u) = 3u2, f(u)x = 6u𝑢𝑥

Let ut + f(u)x = 0

|  |  |
| --- | --- |
|  |  |

Figure (3.1) Note:

1. Forward time (∆t)
2. Backward space (∆x)

ut + f(u)x = 0 → ut = −f(u)x

ui+1,j − ui,j

∆t

f(u)i,j − f(u)i,j−1

= −

∆x

ui+1,j − ui,j

∆t =

f(u)i,j−1 − f(u)i,j

∆x

∆t(f(u)i,j−1 − f(u)i,j) = ∆x(ui+1,j − ui,j) /∆x

∆t

ui+1,j − ui,j = ∆x [f(u)i,j−1 − f(u)i,j]

∆t

ui+1,j = ui,j + ∆x [f(u)i,j−1 − f(u)i,j]

∆x

ui+1,j = ui,j − ∆t [f(u)i,j−1 − f(u)i,j] (3.1)

xi−1 = xi−h, xi+1 = xi+h

From Taylor's:

* 1. yx(i,j) = yx(i)
  2. yx

= yx

− h1 y′(x ) + h2 y′′(x ) − h3 y′′′(x )

(i−1,j)

1. 1!

i 2!

i 3! i

* 1. yx

= yx

− 2h1 y′(x ) + 4h2 y′′(x ) − 8h3 y′′′(x )

(i−2,j)

(i) 1!

i 2!

i 3! i

* 1. yx

= yx

− 3h1 y′(x ) + 9h2 y′′(x ) − 27h3 y′′′(x )

(i−3,j)

(i) 1!

i 2!

i 3! i

Using linear combination:

a(4) + b(3) + c(2) + d(1)

Where:

1. a, b, c, d are constants
2. 1, 2, 3, 4 number of equation

ayx(i−3,j) + byx(i−2,j) + cyx(i−1,j) + dyx(i,j) = (a + b + c +

d)yx

+ (−3a − 2b − c) h1 y′(x ) + (9a + 4b + c) h2 y′′(x ) +

(i,j)

1! i

2! i

(−27a − 8b − c) h3 y′′′(x ) .

3! i

From comparison:

a + b + c + d = 0

−3a − 2b − c = 0 9a + 4b + c = 0

−27a − 8b − c = 0

Solving these equations we get a=-1, b=3, c=-3, d=1.

−yx(i−3,j) + 3yx(i−2,j) − 3yx(i−1,j) + yx(i,j)

y′′′(x) =

h3

# Exact solution of KdV :-

We remember that the simplest mathematical wave is a function of the form 𝑢(𝑥, 𝑡) = 𝑓(𝑥 − 𝑐𝑡)which e.g. is a solution to the simple PDE 𝑢𝑡 +

𝑐𝑢𝑥 = 0where c denotes the speed of the wave. For the well-known wave

equation𝑢𝑡𝑡 − 𝑐2𝑢𝑥𝑥 = 0 the famous d’Alembert solution leads to two wave fronts represented by terms𝑓(𝑥 − 𝑐𝑡) and𝑓(𝑥 + 𝑐𝑡). Hence we start here with atria solution:

u(x, t) = u(x − ct) = u(ξ) (3.2)

ξ = x – ct

∂ξ ∂ξ

∂x = 1 , ∂t = −C

∂u ∂u ∂ξ ∂u

∂x = ∂ξ ∂x =

∂ξ = u′

∂u = ∂u ∂ξ = u′(−c) = −cu′

∂t ∂ξ ∂t

ut(x, t) + 6u(x, t)ux(x, t) + uxxx(x, t) = 0

## -Compensation in equation.

du du d3u

−c dξ + 6u dξ + dξ3 = 0 (3.3)

## -by Integration:-

d2u

−cu + 3u2 +

dξ2

= c1 (3.4)

## Multiple the Eq. (4) by 𝒅𝒖 .

𝒅𝝃

du

−cu dξ

+ 3u2 d u +

dξ

d3u dξ3

d2u

du

= c1 dξ

−cu du + 3u2 du +

dξ2

du = c1du

−cu2 2

+ u3 +

1. du 2
2. (dξ)

du

= c1u + c2 (3.5)

x → ±∞ , u → 0 , dξ = 0

c1 = c2 = 0

−cu2 + 2u3 + (du)2 = 0

dξ

du 2

(dξ)

= cu2 − 2u3 = u2(c − 2u) (3.6)

4 du 𝝃

∫ = ∫ dξ (3.7)

0 √c − 2u 0

u = 1 csech2w (3.8)

2

𝑐 − 2𝑢 = 𝑐(1 − sech2w) = 𝑐 tanh2 w (3.9)

cosh2w − sinh2 = 1

dξ sinh w

dw = −c cosh3w (3.10)

W = sech−1 √2u

c

(3.11)

From (3.7), (3.9), (3.10), (3.11)

−2 w 1

sinh w

ξ = ∫

√c 0

sech2 w. tanh w cosh3w dw

−2 w cosh2w cosh w

sinh w

= ∫

√c 0

sinh w

cosh3w dw

−2 w −2

ξ = ∫ dw = w

√c 0 √c

From (8) we can get:-

−2

ξ =

√𝑐

𝑠𝑒𝑐ℎ−1

2𝑢

√

𝑐

u(ξ) = c sech2 √c ξ

2 2

From eq. (2) we get:-

𝑢(𝑥, 𝑡) = 𝑐 sech2 [√c (x − ct)] (3.12)

2 2

u = − 1 c csch2 w (3.13)

2

u(x, t) = − c csch2 [√c (x − ct)] (3.14)

2 2

eq. (14) is considering irregular solution for (KdV) equation

x

x − ct = 0 → t = c

For my equation (3.12) (3.14) are consider solutions for (KdV) original and easily vary with mathematical calculations

|  |  |
| --- | --- |
|  |  |

Fig 3.4: Solitary wave solution of KdV equation at 𝑡 = 1 𝑠, 𝑐 = 1 𝑚/𝑠

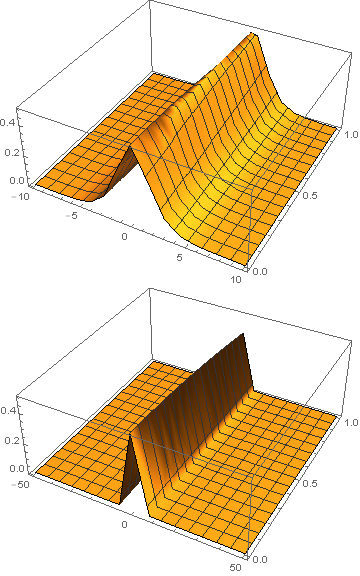


Fig 3.5: One soliton solution of KdV equation at 𝑐 = 1 𝑚/𝑠

# KdV Burger equation:-

## Mathematical Formulation:-

∂u ∂u

𝜕2𝑢

𝜕3𝑢

∂t + 𝛼 u ∂x + 𝛽

𝜕𝑥2 + 𝛾

𝜕𝑥3 = 0 (3.15)

Where α, β, γ are coefficients

Let u(x, 0) = u0(x) (3.16)

# ●Experiential finite difference:-

∂f ∂u

∂u ∂t

= f ′(u). (−𝛼 u ∂u − β

∂x

∂2u

∂x2 − γ

∂3u

∂x3)

2 3

And:− ∂f = f ′ (u). (−𝛼 u ∂u − β ∂ u − γ ∂ u) (3.17)

∂t ∂x

∂x2

∂x3

Using usual forward difference replacement to ∂f we obtain the finite

∂x

difference representation in eq. (3.16)

j+1 j j

∂u ∂2u

∂3u

u1 = f(u1) + kf(u1). (−α u ∂x − β

∂x2 − γ

∂x3)

Let f (u) = ln u, then we obtained the experiential finite difference schema as:-

j+1 j k

j ∂u j

∂2u j

∂3u j

u1 = u1. exp [uj [(−α u1 (∂x)i – β (∂x2)i − γ (∂x3)i)]] (3.18)

(Where k is the time step)

The finite differences for the derivatives have been taken following form:

(∂u j −3ui + 4ui+1 − ui+2

(3.19)

∂x)i = 2h

∂2u j

( ) =

j+1 j+1 j+1 j+1

i i+1 i+2 i+3

2u − 5u + 4u − u

(3.20)

∂x2 i h2

∂3u −5uj+1 + 18uj+1 − 24uj+1 + 14uj+1 − 3uj+1

( )j = i i+1 i+2 i+3 i+4

(3.21)

∂x3 i

∂u

uj+1 − uj+1

2h3

( )j = i+1 i−1

(3.22)

∂x i 2h

∂2u

uj+1 − 2uj+1 + uj+1

( )j = i+1 i i−1

(3.23)

∂x2 i h2

∂3u j

(∂x3 )i

∂u

uj+1−2uj+1+2uj+1−uj+1

=  i+2 i+1 i−1 i−2 . i = 3, . . . , N − 2 (3.24)

2h3

3uj+1 − 4uj+1 + uj+1

( )j = i i−1 i−2

(3.25)

∂x i 2h

∂2u

2uj+1 − 5uj+1 + 4uj+1 − uj+1

( )j = i i−1 i−2 i−3 (12)

∂x2 i h2

∂3u 5uj+1 − 18uj+1 + 24uj+1 − 14uj+1 + 3uj+1

( )j = i i−1 i−2 i−3 i−4

∂x3 i 2h3

, i = N − 1, N . (3.26)

# Classical explicit finite difference method (CEFD):-

∂u j+1

j ∂u j

∂2u j

∂3u j

(∂x)i + α ui(∂x)i + β (∂x2)i + γ (∂x3)i = 0 (3.27)

Then:uj+1 = uj + k

i i

j ∂u j ∂2u j ∂3u j

[−𝛼 ui(∂x)i − 𝛽 (∂x2)i − 𝛾(∂x3)i] (3.28)

Now, we use the equations (3.5) (3.13) in eq. (3.28)

## Analytical Solution:-

Let us take the initial value u(x, 0) as

u(x, 0) =

6β2 25αγ

[1 + tanh (

βx

10γ) +

1

sech2 (

2

βx

10γ)]

u(x, t) =

6β2

25αγ [1

β

+ tanh (

10γ

(x −

6 β2t

25γ ))

1 β

+ sech2 (

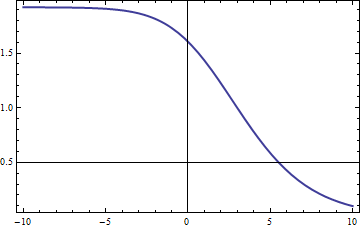
(x −

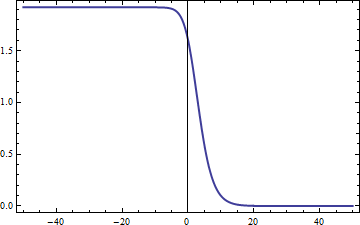
6 β2t

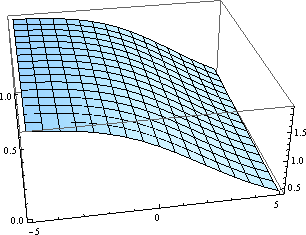
))] (3.29)

2 10γ 25γ

→ u(x, 0) = u0 , x = 0, 1, 2, 3 …







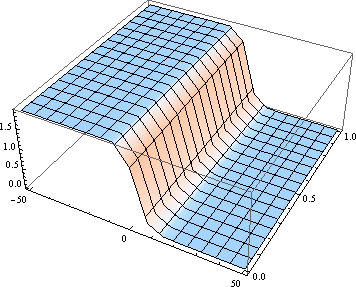


Figure3.6:Kink solution of KdV-B equation.

# Numerical results and discussion:-

1. In this section we present numerical results and compare these with analytical results.
2. We apply the experiential finite difference (CEFD) schemes to non- linear (KdV Burger) equation to compute solutions numerically and compare these solutions with exact solutions at various times.
3. All numerical computation we performed with the space step h=1, 2

… and the time step k=0.0001 and we take parameters: α=1,β=-2,γ=1

1. The results obtained for problem is displayed in table 1-2 for times t=0.002 and t=0.001 respectively according to the results presented here, the experiential finite difference scheme behaves better than the other numerical scheme at small times.
2. We see an excellent agreement of the numerical results with the analytical results.

# Applications of Kortwege-de-Vries:

The KdV equation has several connections to physical problems. In addition to being the governing equation of the string in the [Fermi–Pasta–](https://en.wikipedia.org/wiki/Fermi%E2%80%93Pasta%E2%80%93Ulam%E2%80%93Tsingou_problem) [Ulam–Tsingou problem](https://en.wikipedia.org/wiki/Fermi%E2%80%93Pasta%E2%80%93Ulam%E2%80%93Tsingou_problem) in the continuum limit, it approximately describes the evolution of Long, one-dimensional waves in many physical settings, including:

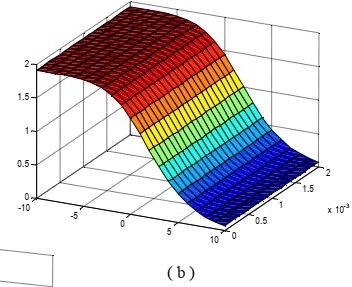
* shallow**-**water waves with weakly [non**-**linear](https://en.wikipedia.org/wiki/Non-linear) restoring forces.
* long [internal waves](https://en.wikipedia.org/wiki/Internal_waves) in a density stratified ocean.
* [ion acoustic waves](https://en.wikipedia.org/wiki/Ion_acoustic_wave) in a [plasma](https://en.wikipedia.org/wiki/Plasma_(physics)).
* [Acoustic](https://en.wikipedia.org/wiki/Acoustics) waves on a [crystal lattice](https://en.wikipedia.org/wiki/Crystal_lattice).

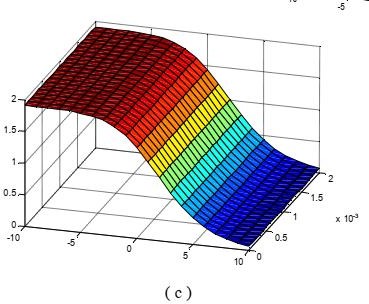
The KdV equation can also be solved using the [inverse scattering](https://en.wikipedia.org/wiki/Inverse_scattering_transform) [transform](https://en.wikipedia.org/wiki/Inverse_scattering_transform) such as those applied to the [non-linear Schrödinger equation](https://en.wikipedia.org/wiki/Non-linear_Schr%C3%B6dinger_equation)***.***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | | |  |  |
|  |  | o **3.6 Comparison between Experiential, CEFD and exact solution of KdV .B equation at t=0.002.** | | | | | |  |  |
|  | | | X | Experiential | CEFD | Exact |  | | |
| -10 | 1.919379002634842 | 1.919379191388729 | 1.919329808148882 |
| -9 | 1.918643308672068 | 1.918644633876643 | 1.918643821458763 |
| -8 | 1.917060591127396 | 1.917065926974663 | 1.917059152453014 |
| -7 | 1.913722954398500 | 1.913721949715588 | 1.919381659478264 |
| -6 | 1.906781239643815 | 1.906781242136646 | 1.912891654544670 |
| -5 | 1.892835885409647 | 1.866026300933741 | 1.906736706456621 |
| -4 | 1.866026301491624 | 1.817448765469159 | 1.892054960158769 |
| -3 | 1.817448766909415 | 1.735914792087203 | 1.865090378129703 |
| -2 | 1.735814795095435 | 1.611343534006028 | 1.817246288454736 |
| -1 | 1.611343538917609 | 1.440592298776634 | 1.735653118557910 |
| 0 | 1.440592304868509 | 1.232342477311477 | 1.606520529450112 |
| 1 | 1.232347317115215 | 1.006948954010265 | 1.440368969120364 |
| 2 | 1.006349957875933 | 0.786237587888046 | 1.232443534579844 |
| 3 | 0.786237589842130 | 0.591000668241789 | 1.006390892145665 |
| 4 | 0.591000668951221 | 0.430510400558712 | 0.786292981214647 |
| 5 | 0.430510400722553 | 0.306113481413995 | 0.591219785154487 |
| 6 | 0.306113481426435 | 0.213810056488724 | 0.432546843425977 |
| 7 | 0.213810056492372 | 0.147446152912332 | 0.213968275257439 |
| 8 | 0.147446152916699 | 0.140446152912322 | 0.147557854727381 |
| 9 | 0.100767547648177 | 0.100767547637227 | 0.100848994270706 |
| 10 | 0.068439291585236 | 0.068489291569168 | 0.068492090192202 |
| 48 | | | | | | | | | |
|  |  |  | | | | | |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | | |  |  |
|  |  | o **3.7 Comparison between Experiential, CEFD and exact solution of KdV .B equation at t=0.001.** | | | | | |  |  |
|  | | | -50 | 1.920000000000000 | 1.920000000000000 | 1.920000000000000 |  | | |
| -48 | 1.920000000000000 | 1.920000000000000 | 1.920000000000000 |
| -46 | 1.920000000000000 | 1.920000000000000 | 1.920000000000000 |
| -44 | 1.919999999999999 | 1.919999999999999 | 1.919999999999999 |
| -42 | 1.919999999999995 | 1.919999999999995 | 1.919999999999995 |
| -40 | 1.919999999999976 | 1.919999999999976 | 1.919999999999976 |
| -38 | 1.919999999999881 | 1.919999999999881 | 1.919999999999881 |
| -36 | 1.919999999999408 | 1.919999999999408 | 1.919999999999408 |
| -34 | 1.919999999997066 | 1.919999999997066 | 1.919999999997066 |
| -32 | 1.919999999985468 | 1.919999999985468 | 1.919999999985468 |
| -30 | 1.999998888928025 | 1.999998888928025 | 1.999998888928025 |
| -28 | 1.9999999643509 | 1.9999999643509 | 1.9999999643509 |
| -26 | 1.99999998234346 | 1.99999998234346 | 1.99999998234346 |
| -24 | 1.9999991265290 | 1.9999991265290 | 1.9999991265290 |
| -22 | 1.9999956694102 | 1.9999956694102 | 1.9999956694102 |
| -20 | 1.9999785580899 | 1.9999785580899 | 1.9999785580899 |
| -18 | 1.9998938816973 | 1.9998938816973 | 1.9998938816973 |
| -16 | 1.9994753189718 | 1.9994753189718 | 1.9994753189718 |
| -14 | 1.9974114126656 | 1.9974114126656 | 1.9974114126656 |
| -12 | 1.9872899009072 | 1.9872899009072 | 1.9872899009072 |
| -10 | 1.9382507871398 | 1.9382507871398 | 1.9382507871398 |
| -8 | 1.9170689275842 | 1.9170689275842 | 1.9170689275842 |
| -6 | 1.9067611361566 | 1.9067611361566 | 1.9067611361566 |
| -4 | 1.86590978644420 | 1.865908786484020 | 1.865908786484020 |
| -2 | 1.73555529627378 | 1.735555298273523 | 1.735555298273523 |
| 49 | | | | | | | | | |
|  |  |  | | | | | |  |  |

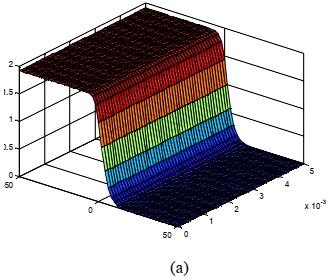
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | | |  |  |
|  |  |  | | | | | |  |  |
|  | 0 | 1.440034892014480 | 1.440034892014480 | 1.440034892014480 |  |
| 2 | 1.005932997922627 | 1.005932997922627 | 1.005932997922627 |
| 4 | 0.590866411570610 | 0.5908664115700610 | 0.590866411570610 |
| 6 | 0.306131865004398 | 0.306131863466041 | 0.30611148463838 |
| - |  |  |  |
| - |  |  |  |
| - |  |  |  |
| - |  |  |  |
| - |  |  |  |
| - |  |  |  |
| 46 | 0.000000039154376 | 0.000000039154347 | 0.000000039203914 |
| 48 | 0.000000017592195 | 0.000000017593182 | 0.000000017615454 |
| 50 | 0.00000007905180 | 0.00000007905126 | 0.00000007915134 |
|  | 50 | | | |  |
|  |  |  | | | | | |  |  |

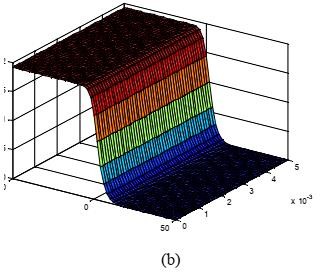


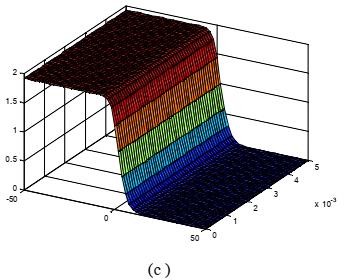


**Figure** 3.7:The numerical results of the KdV-B equation at h=1 by using

:(a) Exponential finite-difference method:(b)classical explicit finite- difference method:(c)The analytical solution .







**Figure** 3.8:The numerical results of the KdV-B equation at h=1 by using

:(a) Exponential finite-difference method:(b)classical explicit finite- difference method:(c)The analytical solution .

# Chapter four

**Computational Programs for Solving Partial Differential Equations 4.1MATLAB**

# MATLAB is widely utilised in all areas of applied mathematics, as well as in university education and research, as well as in industry. The software is constructed around vectors and matrices, and its name stands for matrix laboratory. This makes MATLAB particularly useful for linear algebra, although it may also be used to solve algebraic and differential equations, as well as numerical integration. MATLAB includes sophisticated graphic features and can create beautiful 2D and 3D images. It's also a programming language, and it's one of the most user-friendly for creating mathematical programmes. Signal processing, image processing, optimization, and other tools are available in MATLAB.

# Useful functions and operations in MATLAB

Using MATLAB as a calculator is easy.

*Example:* Compute 5 𝑠𝑖𝑛(2.53 − 𝑝𝑖) + 1/75. In MATLAB this is done by simply typing

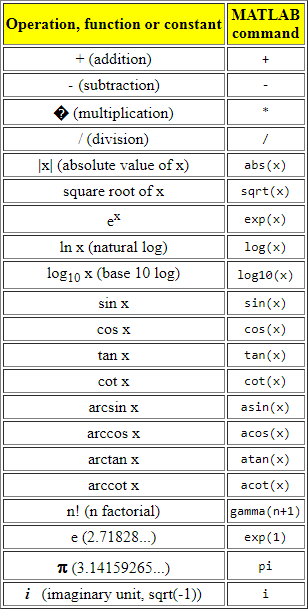
𝟓 ∗ 𝒔𝒊𝒏(𝟐. 𝟓^(𝟑 − 𝒑𝒊)) + 𝟏/𝟕𝟓

at the prompt. Be careful with parantheses and don't forget to type \* whenever you multiply!

Note that MATLAB is *case sensitive*. This means that MATLAB knows a difference between letters written as lower and upper case letters. For

example, MATLAB will understand 𝒔𝒊𝒏(𝟐) but will not understand 𝑺𝒊𝒏(𝟐).

Here is a table of useful operations, functions and constants in MATLAB.



# Obtaining Help on MATLAB commands

To obtain help on any of the MATLAB commands, you simply need to type

# help<command>

at the command prompt. For example, to obtain help on the gamma function, we type at the command prompt:

# help gamma

Try it right now. You may also seek help with commands by going to the "Help Desk," which you can find by going to the Help menu and selecting the MATLAB Help option.

Note that the command name in ALL CAPS appears in the description MATLAB delivers about the command you requested help on. This does not imply that you must type the command in ALL CAPS. When using a command in MATLAB, you almost usually use all lower case letters.

# Variables in MATLAB

We can easily define our own variables in MATLAB. Let's say we need to use the value of 3.5𝑠𝑖𝑛(2.9) repeatedly. Instead of typing 3.5 ∗

𝑠𝑖𝑛(2.9)over and over again, we can denote this variable as x by typing the following:

𝒙 = 𝟑. 𝟓 ∗ 𝒔𝒊𝒏(𝟐. 𝟗)

(Please try this in MATLAB.) Now type

𝒙 + 𝟏

and observe what happens. Note that we did not need to declare x as a variable that is supposed to hold a floating point number as we would need to do in most programming languages.

Often, we may not want to have the result of a calculation printed-out to the command window. To supress this output, we put a semi-colon at the end of the command; MATLAB still performs the command in "the background". If you defined x as above, now type

𝒚 = 𝟐 ∗ 𝒙;

𝒚

and observe what happened.

In many cases we want to know what variables we have declared. We can do this by typing **whos**. Alternatively, we can view the values by openning the "Workspace" window. This is done by selecting the Workspace option from the View menu. If you want to erase all variables from the MATLAB memory, type **clear.**To erase a specific variable, **say x**, type **clear x**. To clear two specific variables, say x and y, type **clear x y**, that is separate the different variables with a space. Variables can also be cleared by selecting them in the Workspace window and selecting the delete option.

# How to plot with MATLAB

There are different ways of plotting in MATLAB. The following two techniques, illustrated by examples, are probably the most useful ones.

*Example 1:* Plot 𝒔𝒊𝒏(𝒙𝟐) on the interval [-5,5]. To do this, type the following:

# x=-5:0.01:5;

**y=sin(x.^2);**

# plot(x,y)

and observe what happens.

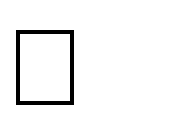
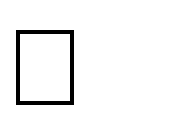
*Example 2:* Plot **exp(sin(x))** on the interval [- following:



, ].

# x=linspace(-pi,pi,101); y=exp(sin(x)); plot(x,y)

To do this, type the

And observe what happens. The command linear space creates a vector of 101 equally spaced values between **-** and (inclusive).

Occasionally, we need to plot values that vary quite differently in magnitude. In this case, the regular plot command fails to give us an adequate graphical picture of our data. Instead, we need a command that plots values on a log scale. MATLAB has 3 such commands: **loglog,semilogx**, and **semilogy.** Use the help command to see a description of each function. As an example of where we may want to use one of these plotting routines, consider the following problem:

*Example 3:* Plot **x5/2** for **x = 10-5** to **105**. To do this, type the following:

# x=logspace(-5,5,101); y=x.^(5/2);

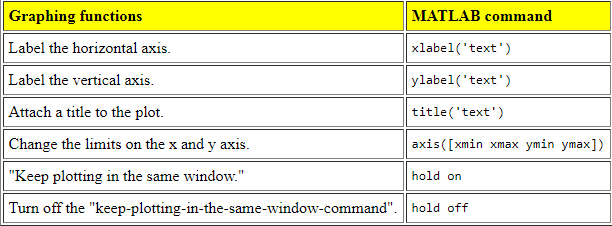
**plot(x,y)**

and observe what happens. Now type the following command:

# loglog(x,y)

The command **logspace** is similar to **linspace**, however it creates a vector of 101 points lograthmically equally distributed between **10-5** and **105**.

The following commands are useful when plotting:



Note that all text must be put within ' '. The last two commands (hold on and hold off) are best explained by trying them next time you plot.

* 1. **Computational programs for solving PDE by MATLAB**

clc clearall; closeall;

% Iterative Solutions of linear equations: Jacobi Method

% Linear system: A u = B

% Coefficient matrix A, right-hand side vector B A=[4 -1 -1 0; -1 4 0 -1; -1 0 4 -1; 0 -1 -1 4];

B= [1/3;4/3;0;1/3];

% Set initial value of u to zero column vector u0=zeros(1,4);

%u1=0;

% Set Maximum iteration number k\_max k\_max=6;

% Set the convergence control parameter erp erp=0.0001;

% Show the q matrix

% loop for iterations for k=1:k\_max

for i=1:4 s=0.0;

for j=1:4 if j==i continue else

s=s+A(i,j)\*u0(j); end

end

u1(i)=(B(i)-s)/A(i,i); end

if norm(u1-u0)<erp break

else u0=u1; end end

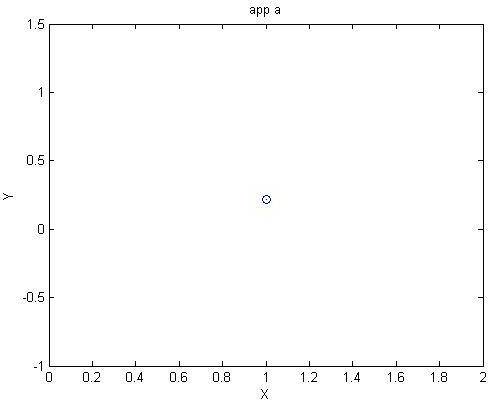
% show the final solution u=u1

% show the total iteration number n\_iteration=k

plot(u(:,1),'-o')

xlabel('X');ylabel('Y'); title('app a');

% figure;



Figure(4.1)

clc clearall; closeall;

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

% error tolerance tol = 0.0001;

%initial guess:

x0 = zeros(4,1);

% Jacobi method

% xnew=x0; error=1;

while error>tol xold=xnew;

for i=1:length(xnew)

off\_diag = [1:i-1 i+1:length(xnew)];

xnew(i) = 1/A(i,i)\*( b(i)-sum(A(i,off\_diag)\*xold(off\_diag)) ); end

error=norm(xnew-xold)/norm(xnew); end

x\_jacobian=xnew;

% maxiter=6; lambda=1; n=length(x0); x=x0;

error=1; iter = 0;

while (error>tol&iter<maxiter) xold=x;

for i=1:n

I = [1:i-1 i+1:n];

x(i) = (1-lambda)\*x(i)+lambda/A(i,i)\*( b(i)-A(i,I)\*x(I) ); end

error = norm(x-xold)/norm(x); iter = iter+1;

end x\_siedal=x

%SOR

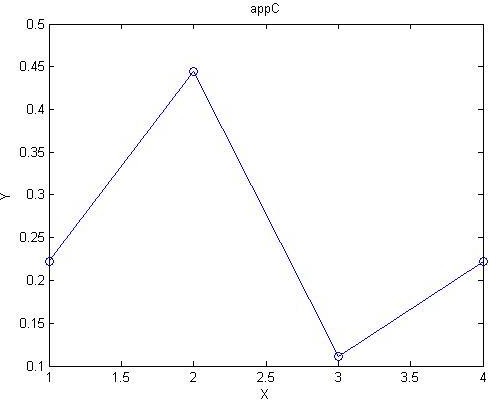
% lambda=1.3; n=length(x0);

x=x0; error=1; iter = 0;

plot(xnew(:,1),'-o')

xlabel('X');ylabel('Y'); title('appC');

figure;



Figure(4.2)

%app d clc clearall; closeall;

% Matlab code for conjugate gradient method.

% function [u, niter, flag] = solveCG(A, f, s, tol, maxiter)

% SOLVECG Conjugate Gradients method.

%

% Input parameters:

% A : Symmetric, positive definite NxN matrix

% f : Right-hand side Nx1 column vector

% s : Nx1 start vector (the initial guess)

% tol : relative residual error tolerance for break

% condition

% maxiter : Maximum number of iterations to perform

%

% Output parameters:

% u : Nx1 solution vector

% niter : Number of iterations performed

% flag : 1 if convergence criteria specified by TOL could

% not be fulfilled within the specified maximum

% number of iterations, 0 otherwise (= iteration

% successful).

A=[4 -1 -1 0; -1 4 0 -1; -1 0 4 -1; 0 -1 -1 4] f=[1/3;4/3;0;1/3]

s=[0;0;0;0]

maxiter = 6;

u = s; % Set u\_0 to the start vector s r = f - A\*s % Compute first residuum p = r;

rho = r'\*r

niter = 0; % Init counter for number of iterations flag = 0; % Init break flag

% Compute norm of right-hand side to take relative residuum as

% break condition. normf = norm(f);

ifnormf<eps% if the norm is very close to zero, take the

% absolute residuum instead as break condition

% ( norm(r) >tol ), since the relative

% residuum will not work (division by zero).

warning(['norm(f) is very close to zero, taking absolute residuum'... ' as break condition.']);

normf = 1;

end

while (norm(r)/normf> 0.00001) % Test break condition a = A\*p;

alpha = rho/(a'\*p); u = u + alpha\*p;

r = r - alpha\*a; rho\_new = r'\*r;

p = r + rho\_new/rho \* p; rho = rho\_new; niter = niter + 1;

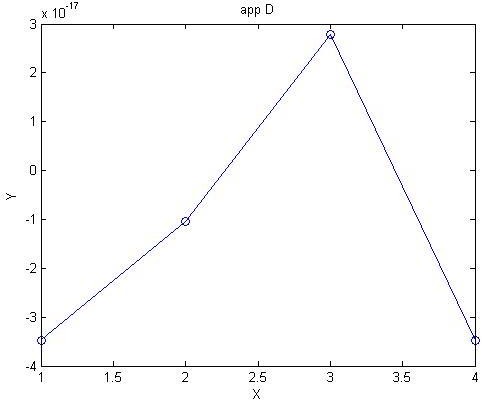
if (niter == maxiter) % if max. number of iterations flag = 1; % is reached, break.

break end end

plot(p(:,1),'-o')

xlabel('X');ylabel('Y'); title('app D');

figure;



Figure(4.3)

% Matlab code for Jacobi method.

% Iterative Solutions of linear equations: Jacobi Method

% Linear system: A x = B

% Coefficient matrix A, right-hand side vector B clc

clearall

A=[7/2 -2 0 -1 0 0; -1 4 -1 0 -1 0; 0 -1 4 0 0 -1; -1 0 0 7/2 -2 0; 0 -1 0 -1 4

-1; 0 0 -1 0 -1 4];

B= [29/16; 5/8; 23/16; 11/8; 3/4; 13/8];

% Set initial value of x to zero column vector x0=zeros(1,6);

% Set Maximum iteration number k\_max k\_max=28;

% Set the convergence control parameter erp erp=0.0001;

% Show the q matrix

% loop for iterations for k=1:k\_max

for i=1:6 s=0.0;

for j=1:6 if j==i continue else

s=s+A(i,j)\*x0(j); end

end

x1(i)=(B(i)-s)/A(i,i); end

if norm(x1-x0)<erp break

else x0=x1; end end

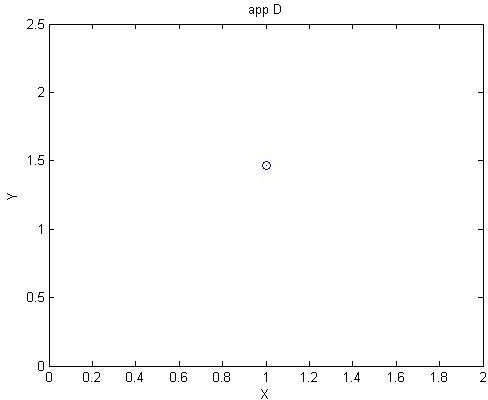
% show the final solution x=x1

% show the total iteration number n\_iteration=k

plot(x(:,1),'-o')

xlabel('X');ylabel('Y'); title('app D');

figure;



Figure(4.4)

%Matlab code for SOR method. clc

closeall; clearall;

A = [7/2 -2 0 -1 0 0; -1 4 -1 0 -1 0; 0 -1 4 0 0 -1; -1 0 0 7/2 -2 0; 0 -1 0 -1

4 -1; 0 0 -1 0 -1 4]; % coefficients matrix b = [29/16; 5/8; 23/16; 11/8; 3/4; 13/8];

% error tolerance tol = 0.0001;

%initial guess:

x0 = zeros(6,1);

% Jacobi method

% xnew=x0; error=1;

while error>tol xold=xnew;

for i=1:length(xnew) off\_diag = [1:i-1 i+1:length(xnew)]; xnew(i) = 1/A(i,i)\*( b(i)-sum(A(i,off\_diag)\*xold(off\_diag)) ); end

error=norm(xnew-xold)/norm(xnew); end

x\_jacobian=xnew;

%Gauss?Seidel:

% maxiter=10; lambda=1; n=length(x0); x=x0;

error=1; iter = 0;

while (error>tol&iter<maxiter) xold=x;

for i=1:n

I = [1:i-1 i+1:n]; x(i) = (1-lambda)\*x(i)+lambda/A(i,i)\*( b(i)-A(i,I)\*x(I)

);

end

error = norm(x-xold)/norm(x); iter = iter+1;

end x\_siedal=x

%SOR

%

lambda=1.3; n=length(x0); x=x0; error=1;

iter = 0;

while (error>tol&iter<maxiter) xold=x;

for i=1:n

I = [1:i-1 i+1:n];

x(i) = (1-lambda)\*x(i)+lambda/A(i,i)\*( b(i)-A(i,I)\*x(I) ); end error = norm(x-xold)/norm(x);

iter = iter+1; end x\_SOR=x

plot(x(:,1),'-o')

xlabel('X');ylabel('Y'); title('app G');

figure;

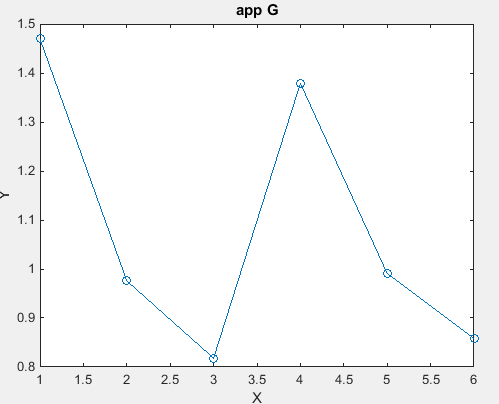
% plot(x(:,1),'-',x(:,2),'-.')

% xlabel('X');

% ylabel('Y');

% legend('y');

% title('Comparison Between sidle and sor ');



Figure(4.6)

%Matlab code for Conjugate Gradient method. function [u, niter, flag] = solveCG(A, f, s, tol, maxiter)

% SOLVECG Conjugate Gradients method.

%

% Input parameters:

% A : Symmetric, positive definite NxN matrix

% f : Right-hand side Nx1 column vector

% s : Nx1 start vector (the initial guess)

% tol : relative residual error tolerance for break

% condition

% maxiter : Maximum number of iterations to perform

%

% Output parameters:

% u : Nx1 solution vector

% niter : Number of iterations performed

% flag : 1 if convergence criteria specified by TOL could

% not be fulfilled within the specified maximum

% number of iterations, 0 otherwise (= iteration

% successful). clc

clearall

A=[7/2 -2 0 -1 0 0; -1 4 -1 0 -1 0; 0 -1 4 0 0 -1; -1 0 0 7/2 -2 0; 0 -1 0 -1 4

-1; 0 0 -1 0 -1 4];

f=[29/16; 5/8; 23/16; 11/8; 3/4; 13/8]; s=[0;0;0;0;0;0] ;

maxiter = 6;

u = s; % Set u\_0 to the start vector s

r = f - A\*s; % Compute first residuum p = r;

rho = r'\*r;

niter = 0; % Init counter for number of iterations flag = 0; % Init break flag

% Compute norm of right-hand side to take relative residuum as

% break condition. normf = norm(f);

ifnormf<eps% if the norm is very close to zero, take the

% absolute residuum instead as break condition

% ( norm(r) >tol ), since the relative

% residuum will not work (division by zero).

warning(['norm(f) is very close to zero, taking absolute residuum'... ' as break condition.']);

normf = 1; end

while (norm(r)/normf> 0.00001) % Test break condition a = A\*p;

alpha = rho/(a'\*p); u = u + alpha\*p;

r = r - alpha\*a;

rho\_new = r'\*r; p = r + rho\_new/rho \* p; rho = rho\_new;

niter = niter + 1;

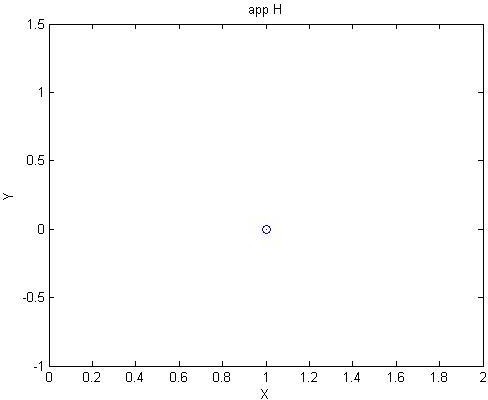
if (niter == maxiter) % if max. number of iterations flag = 1; % is reached, break.

break end end

plot(rho(:,1),'-o')

xlabel('X');ylabel('Y'); title('app H');

figure;



Figure(4.7)

% KdV.m - Solve KdV equation by Fourier spectral/ETDRK4 scheme

% A.-K.Kassam and L. N. Trefethen 4/03

% This code solves the Korteweg-de Vries eq. u\_t+uu\_x+u\_xxx=0

% with periodic BCs on [-pi,pi] and initial condition given by

% a pair of solitons. The curve evolves up to t=0.005 and at

% the end u(x=0) is printed to 6-digit accuracy. Changing N

% to 384 and h to 2.5e-7 improves this to 10 digits but takes

% four times longer. clear

clc

% Set up grid and two-soliton initial data:

N = 512;

x = (2\*pi/N)\*(-N/2:N/2-1)'; A = 25; B = 16;

u = 3\*A^2\*sech(.5\*(A\*(x+2))).^2+3\*B^2\*sech(.5\*(B\*(x+1))).^2; p = plot(x,u,'linewidth',3);

axis([-pi pi -200 2200]), grid on

% Precompute ETDRK4 scalar quantities (Kassam-Trefethen): h = 1e-6; % time step

k = [0:N/2-1 0 -N/2+1:-1]'; % wave numbers L = 1i\*k.^3; % Fourier multipliers

E = exp(h\*L); E2 = exp(h\*L/2);

M = 64; % no. pts for complex means

r = exp(2i\*pi\*((1:M)-0.5)/M); % roots of unity LR = h\*L(:,ones(M,1))+r(ones(N,1),:);

Q = h\*mean( (exp(LR/2)-1)./LR ,2);

f1 = h\*mean((-4-LR+exp(LR).\*(4-3\*LR+LR.^2))./LR.^3,2);

f2 = h\*mean( (4+2\*LR+exp(LR).\*(-4+2\*LR))./LR.^3,2);

f3 = h\*mean((-4-3\*LR-LR.^2+exp(LR).\*(4-LR))./LR.^3,2);

g = -.5i\*k;

% Time-stepping by ETDRK4 formula (Cox-Matthews): set(gcf,'doublebuffer','on')

disp('press <return> to begin'), pause % wait for user input t = 0; step = 0; v = fft(u);

whilet+h/2 < 0.006 step = step+1;

t = t+h;

Nv = g.\*fft(real(ifft(v)).^2);

a = E2.\*v+Q.\*Nv; Na = g.\*fft(real(ifft(a)).^2); b = E2.\*v+Q.\*Na; Nb = g.\*fft(real(ifft(b)).^2);

c = E2.a+Q.(2\*Nb-Nv); Nc = g.\*fft(real(ifft(c)).^2); v = E.\*v+(Nv.\*f1+(Na+Nb).\*f2+Nc.\*f3);

if mod(step,25)==0

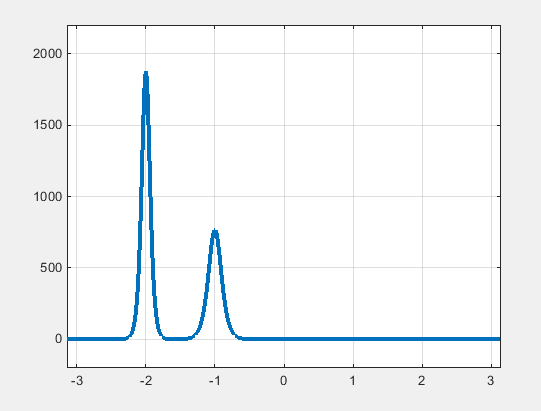
u = real(ifft(v)); set(p,'ydata',u)

title(sprintf('t = %7.5f',t),'fontsize',18), drawnow end

end

text(-2.4,900,sprintf('u(0) = %11.7f',u(N/2+1)),... 'fontsize',18,'color','r')

Figure;



Figure(4.8)

# An introduction to Mathematica

Mathematica is a massive and ostensibly complicated programme. It includes hundreds of functions for completing activities in science, mathematics, and engineering, such as computing, programming, data analysis, knowledge representation, and data visualisation. Of this chapter, we cover the fundamental operations in Mathematica and give an overview of its computational and programming capabilities. Furthermore, we include some basic facts that Mathematica users should be aware of, such as how to start Mathematica, exit it, enter simple inputs and obtain results, and ultimately, how to use Mathematica's documentation to obtain answers to problems regarding the system.

## A brief overview of Mathematica Numerical computations

*Mathematica* has been aptly described as a sophisticated calculator. With it you can enter mathematical expressions and compute their values

𝐼𝑛[1] = Sin[. 86] − 𝐿𝑜𝑔[𝜋](1 + . 8 )12

12

𝑂𝑢𝑡[1] = −0.481899

You can store values in memory.

*In[2] =* **rent= 350**

*Out[2]=* 350

*In[3]:=* **food= 175**

*Out[3]=* 175

*In[4]:=* **heat= 83**

*Out[4]=* 83

*In[5]:=* **rent+ food+ heat**

*Out[5]=* 608

Yet *Mathematica* differs from calculators and simple computer programs in its ability to

calculate exact results and to compute to an arbitrary degree of precision.

𝐼𝑛[6] = 1

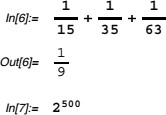
15

+ 1 + 1

35 63

𝑂𝑢𝑡[6] = 1

9



1 9

𝑰𝒏[𝟕] = 𝟐𝟓𝟎𝟎

*Out[7]=*3273390607896141870013189696827599152216642046043064 78948329136809613379640467455488327009232590415715088668412

7560071009217256545885393053328527589376

## Symbolic computation

One of the more powerful features of *Mathematica* is its ability to manipulate and compute

with symbolic expressions. For example, you can factor polynomials and simplify trigonometric expressions.

## Graphics

𝐼𝑛[8] = 𝐹𝑎𝑐𝑡𝑜𝑟[𝑥5 − 1]

𝑂𝑢𝑡[8] = (−1 + 𝑥)(1 + 𝑥 + 𝑥2 + 𝑥3 + 𝑥4)

𝐼𝑛[9] = 𝑇𝑟𝑖𝑔𝑅𝑒𝑑𝑢𝑐𝑒[𝑆𝑖𝑛[𝜃]3]

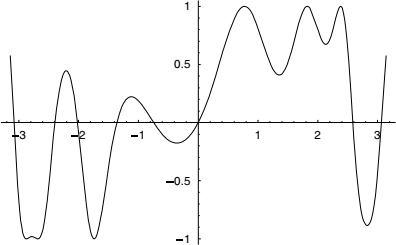
𝑂𝑢𝑡[9] = 1(3 sin[𝜃] − sin[3𝜃])

4

The ability to visualize functions or sets of data often allows us greater insight into their structure and properties. *Mathematica* provides a wide range of graphing capabilities. These include two- and three-dimensional plots of functions or data sets, contour and density plots of functions of two variables, bar charts, histograms and pie charts of data sets, and many packages designed for specific graphical purposes. In addition, the *Mathematica* programming language allows you to construct graphical images “from the ground up” using primitive elements

Here is a simple two-dimensional plot of The function sin( 𝑥 + √2 sin(𝑥2)

𝐼𝑛[10] = 𝑃𝑙𝑜𝑡[sin[𝑥 + √2 sin[𝑥2]], {𝑥, −𝜋, 𝜋}



𝑂𝑢𝑡[10] = −𝐺𝑟𝑎𝑝ℎ𝑖𝑐𝑠

Here is a surface of constant negative curvature, represented parametrically by the three functions 𝛒, and𝑟. This surface is often referred to as Dini's surface

