

REPORT ON NUMERICAL ANALYSIS

Avrojit Joydhar(IMS23068)

May 2024 - April 2025

Contents

1	Introduction	3
1.1	Solution to Ordinary Differential Equations	3
1.2	Solution to Partial Differential Equations	4
1.3	Why do we need Numerical Analysis?	4
2	Numerical Solutions of Ordinary Differential Equations	6
2.1	IVP VS BVP	6
2.2	Single Step Methods vs Multi Step Methods	7
2.3	Picard's Method	7
2.4	Euler's Method	7
2.4.1	Problems with Euler's Method	8
2.5	Modified Euler's Method	9
2.6	Runge Kutta Methods	10
2.6.1	RK2	11
2.6.2	RK3	14
2.6.3	RK4	14
2.7	Adam's Predictor and Corrector Methods	17
3	Order of Convergence	20
3.1	Theory	20
3.1.1	Local Truncation Error	20
3.1.2	Global Error	21
3.2	Order of Convergence of Some Numerical Methods	21
3.2.1	Picard's Method	21
3.2.2	Runge Kutta Method Of 2nd Order	22
3.2.3	Runge Kutta Method of Order 4	24
4	Partial Differential Equations	26
4.0.1	Linearity	26
4.1	First Order Equations	27
4.1.1	Solving Methods	27
4.2	Types of First-Order Differential Equations	27
4.2.1	Separable Equations	28
4.2.2	Linear Equations	28

4.2.3	Exact Equations	28
4.2.4	Homogeneous Equations	28
4.2.5	Bernoulli Equations	28
4.3	Second Order Equations	29
4.4	The Heat Equation	31
4.4.1	Uniqueness	31
4.4.2	Stability	32
4.4.3	Diffusion on the Whole Line	32
4.5	Reflections of Waves	33
5	Boundary Value Problems	35
5.1	The Dirichlet Condition	35
5.2	The Neumann Boundary	37
5.3	The Robin Condition	37
6	Fourier Series	39
6.1	The Complex Form	40
6.2	Back to the real form	41
6.3	Magnitude-angle form	42
7	Finite Element Methods	43
7.1	Introduction	43
7.2	The Galerkin Method	43
7.2.1	L-p Spaces	44
7.2.2	Sobolev Spaces	44
7.2.3	Weak Formulation of Sobolev Spaces	45
7.2.4	Coming back to Galerkin Method	46
7.3	Shape Functions	47
7.4	Solving 1D Poisson Equation using the Galerkin Method in C++ and Gnuplot	48
7.5	MESH	50
7.5.1	Types of Meshes	51
7.5.2	Simple Mesh Implementation using Python and Matplotlib	51
8	References	54

Chapter 1

Introduction

A number of scientific and technological issues can be very easily translated to differential equations. Furthermore, the analysis of differential equations would provide us with solutions which would benefit rather even aid such issues in a more delicate manner than one could think of. Quite often differential equations appearing in physical problems do not belong to any of these familiar types and one is obliged to resort to numerical methods. These methods are of even greater importance when we realize that computing machines are now readily available which reduce numerical work considerably.

The main aim of differential equations are to study the solutions that satisfy the equations and the properties of the solutions.

Suppose a population equation, which looks like

$$p(t) = t^2 + 10t + 50$$

The purpose of this function would be that: you can input a time t and it would show us the population size at that time. What we mean when it comes to finding the solution of it would be the value of t when $p(t) = 0$ where p could be any scalar quantity.

1.1 Solution to Ordinary Differential Equations

The solution of an ordinary differential equation means finding an explicit expression for y in terms of a finite number of elementary functions of x . Such a solution of a differential equation is known as the **closed or finite form** of solution.

$$\frac{dy}{dx} = f(x, y), \text{ given } y(x_0) = y_0 \quad (1.1)$$

Here there are different methods to solve a differential equations like this. We have multiple variety of methods and for each type of differential equation.

These methods are some kind of tricks in an algorithmic manner which one should follow to solve the equations. There is the Variable Separable Method for simple differential equations, specific methods to treat Homogenous Differential Equations or Linear Differential Equations. One of the most efficient ways of such methods would be replacing the differential equation to a **difference equation** and then solve it.

1.2 Solution to Partial Differential Equations

Partial differential equations (PDEs) are a fundamental tool for modeling and analyzing complex phenomenas. One common approach ,as we have seen earlier in differential equations to use the method of separation of variables, which involves expressing the solution as a product of functions of space and time. In linear Partial Differential Equations where Seperation of Variables is found to be very effective especially in the heat equations.

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

As we can see the equation is a complete statement. It shows how the temperature evolves along the rod. Solving such equations is comparatively easier but many PDEs do not have closed-form analytical solutions, especially for complex geometries, nonlinear problems, or high-dimensional problems. Also for simulating environments solving such equations are of no use because the efficiency at which it takes time to process these equations is very slow.

Here the idea of Numerical Analysis of Differential equations comes to play.

1.3 Why do we need Numerical Analysis?

Computational power and efficiency: Numerical methods can be implemented on computers, allowing for rapid and efficient computation of solutions. This is particularly important for large-scale problems or those requiring repeated simulations.

Flexibility and adaptability: Numerical methods can be tailored to specific problems, allowing for adjustments to the discretization, time-stepping, and other parameters to optimize the solution.

Error control and estimation: Numerical methods provide a way to estimate the error in the solution, allowing for adaptive refinement and improved accuracy.

Visualization and interpretation: Numerical solutions can be visualized and interpreted more easily than analytical solutions, making it easier to understand the behavior of the system.

Validation and verification: Numerical methods can be used to validate and verify the accuracy of analytical solutions, and vice versa.

Real-world applications: Many real-world problems involve complex geometries, nonlinear interactions, and high-dimensional spaces, making numerical methods essential for solving these problems.

Scalability: Numerical methods can be scaled up to solve large-scale problems, such as those involving millions of degrees of freedom.

Multiscale problems: Numerical methods can be used to solve problems involving multiple scales, such as those involving both macroscopic and microscopic phenomena.

Uncertainty quantification: Numerical methods can be used to quantify uncertainty in the solution, allowing for probabilistic analysis and risk assessment.

Chapter 2

Numerical Solutions of Ordinary Differential Equations

2.1 IVP VS BVP

Initial Value Problems (IVP): Initial value problem does not require to specify the value at boundaries, instead it needs the value during initial condition. This usually apply for dynamic system that is changing over time as in Physics. An example, to solve a particle position under differential equation, we need the initial position and also initial velocity. Without these initial values, we cannot determine the final position from the equation.

Boundary Value Problems (BVP): In contrast, boundary value problems not necessarily used for dynamic system. Instead, it is very useful for a system that has space boundary. An example would be shape from shading problem in computer vision. To determine surface gradient from the ODEs/PDEs, one should impose boundary values on the region of interest. Given the diagram below, the black and red circles represent given boundary and initial values, respectively. The open circles represent the unknown solution.

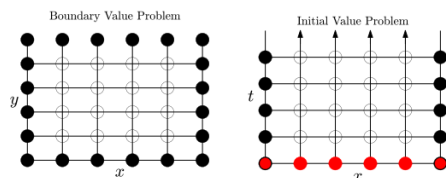


Figure 2.1: A SCHEMATIC DIAGRAM SHOWING THE DIFFERENCE BETWEEN IVP VS BVP

2.2 Single Step Methods vs Multi Step Methods

Single Step Methods: These methods yield solutions either as a power series in x from which the values of y can be found by direct substitution, or a set of values of x and y . The methods of Picard and Taylor series belong to the former class of solutions. In these methods, y in (1) is approximated by a truncated series, each term of which is a function of x . The information about the curve at one point is utilized and the solution is not iterated. As such, these are referred to as single-step methods

Multi Step Methods: The methods of Euler, Runge-Kutta, Milne, Adams-Bashforth, etc. belong to the latter class of solutions. In these methods, the next point on the curve is evaluated in short steps ahead, by performing iterations until sufficient accuracy is achieved. As such, these methods are called step-by-step methods.

2.3 Picard's Method

Picard's Method uses the power of Iteration to find solutions to Initial Value Problems.

Given an equation of the first order

$$\frac{dy}{dx} = f(x, y) \quad (2.1)$$

$$\text{Initial condition being : } y = y_0 \quad (2.2)$$

$$\begin{aligned} y_1 &= y_0 + \int_x^{x_0} f(x_0, y_0) \\ y_2 &= y_0 + \int_x^{x_0} f(x_0, y_1) \\ &\vdots \\ y_{n+1} &= y_0 + \int_x^{x_0} f(x_0, y_n) \end{aligned} \quad (2.3)$$

2.4 Euler's Method

Euler's method is a numerical technique for approximating solutions to ordinary differential equations (ODEs). It involves dividing the time interval into small steps, approximating the derivative at each step, and updating the solution using the approximate derivative.

Let us divide LM into n sub-intervals each of width h at L_1, L_2 so that, h

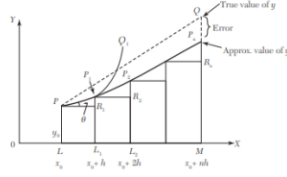


Figure 2.2: Graphical representation of a curve

is quite small. In the interval LL_1 , we approximate the curve by the tangent at P . If the ordinate through L_1 meets this tangent in $P_1(x_0 + h, y_1)$, then

$$\begin{aligned} y_1 &= L_1P_1 = LP + R_1P_1 = y_0 + PR_1 \tan \theta \\ &= y_0 + h \left(\frac{dy}{dx} \right) = y_0 + hf(x_0, y_0) \end{aligned}$$

Let P_1Q_1 be the curve of solution of (1) through P_1 and let its tangent at P_1 meet the ordinate through L_2 in $P_2(x_0 + 2h, y_2)$. Then

$$y_2 = y_1 + hf(x_0 + h, y_1) \quad (2.4)$$

Repeating this process n times, we finally reach on an approximation MP_n of MQ given by

$$y_n = y_{n-1} + hf(x_0 + (n-1)h, y_1) \quad (2.5)$$

This is Euler's method of finding an approximate solution of (2.4).

2.4.1 Problems with Euler's Method

Local truncation error: Euler's method's local truncation error is bounded by a third derivative, which makes it less accurate than higher-order techniques.

Approximating the integral curve: Euler's method approximates the integral curve by a tangent line, which can lead to errors. For example, if the true solution's graph is concave up, the tangent line approximation will be below the true graph and underestimate the solution. If the graph is concave down, the approximation will be above the true graph and overestimate the solution. These under- and overestimates can build up over multiple steps, resulting in significant differences between the approximate and actual values.

Only good for general trends: Euler's method is best for getting general trends and long-term behavior of solutions, rather than providing precise values

2.5 Modified Euler's Method

The Modified Euler's method is also called the midpoint approximation. This method reevaluates the slope throughout the approximation. Instead of taking approximations with slopes provided in the function, this method attempts to calculate more accurate approximations by calculating slopes halfway through the line segment. In Euler's method, the curve of solution in the interval LL_1 is approximated by the tangent at P .

$$x_{n+1} = x_n + \frac{\Delta t}{2} [f(t_n, x_n) + f(t_n + \Delta t, x_{n+1}^*)] \quad (2.6)$$

The obvious problem with this formula is that the unknown value x_{n+1}^* appears on the right-hand-side. We can, however, estimate this value, in what is called the predictor step. For the predictor step, we use the Euler method to find

$$x_{n+1}^* = x_n + \Delta t f(t_n, x_n)$$

If we look closely,

$$k_1 = \Delta t f(t_n, x_n)$$

$$k_2 = \Delta t f(t_n + \Delta t, x_{n+1}^*)$$

$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2) \quad (2.7)$$

```
#include <iostream>
#include <cmath>
#include <functional>

void modifiedEuler(double y0, double x0, double xEnd, double h,
    std::function<double(double, double)> f) {
    double y = y0;
    double x = x0;

    while (x < xEnd) {
        double k1 = f(x, y);
        double k2 = f(x + h, y + h * k1);

        y = y + (h / 2) * (k1 + k2);
        x = x + h;

        std::cout << "x: " << x << ", y: " << y << std::endl;
    }
}

int main() {
    auto f = [](double x, double y) { return std::sin(2 * x); };
```

```

x: 0.1, y: 0.00993347
x: 0.2, y: 0.0393379
x: 0.3, y: 0.0870409
x: 0.4, y: 0.151141
x: 0.5, y: 0.229082
x: 0.6, y: 0.317758
x: 0.7, y: 0.413632
x: 0.8, y: 0.512883
x: 0.9, y: 0.611554
x: 1, y: 0.705712
x: 1.1, y: 0.791601

```

Figure 2.3: Modified Euler Values

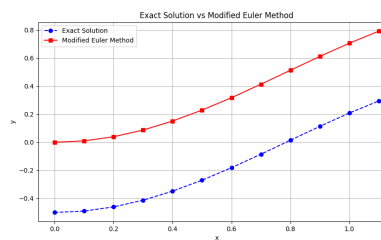


Figure 2.4: Exact Solutions vs Modified Euler Graph

```

double y0 = 0.0;
double x0 = 0.0;
double xEnd = 1.0;
double h = 0.1;

modifiedEuler(y0, x0, xEnd, h, f);

return 0;
}

```

As we can see the Modified Euler Equation is a RUNGE KUTTA METHOD OF ORDER 2.

2.6 Runge Kutta Methods

Runge-Kutta methods are a class of methods which judiciously uses the information on the 'slope' at more than one point to extrapolate the solution to the future time step

2.6.1 RK2

$$\begin{aligned}\frac{dx}{dt} &= f(t_n, x_n) \\ k_1 &= \Delta t(f(t_n, x_n)) \\ k_2 &= \Delta t f(t_n + \alpha, x_{n+1})\end{aligned}\tag{2.8}$$

$$x_{n+1} = x_n + ak_1 + bk_2\tag{2.9}$$

Using Taylor's approximation,

$$\begin{aligned}x_{n+1} &= x_n + \left. \frac{dx}{dt} \right|_{x_i, y_i} (x_{i+1} - x_i) + \frac{1}{2!} \left. \frac{d^2x}{dt^2} \right|_{x_i, y_i} (x_{i+1} - x_i)^2 + \frac{1}{3!} \left. \frac{d^3x}{dt^3} \right|_{x_i, y_i} (x_{i+1} - x_i)^3 + \dots \\ x_{n+1} &= x_n + f(t_n, y_n)h + \frac{1}{2!} f'(t_n, x_n)h^2 + \mathcal{O}(h^2)\end{aligned}\tag{2.10}$$

$\mathcal{O}(h^2)$ means “terms containing second and higher powers of h” and is read as order of h^2

Now putting the comparing the values of k_1 and k_2 in equation 3.2, we get:

$$a + b = 1$$

$$\alpha a = 1/2$$

$$\beta b = 1/2$$

So RK2 looks like :

$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2)$$

```
#include <iostream>
#include <cmath>
#include <limits>

double dydx(double x, double y) {
    return std::log(x * y);
}

void rungeKutta2(double x0, double y0, double x, double h) {
    double k1, k2;
    double y = y0;
    double x_curr = x0;

    while (x_curr < x) {
        if (x_curr + h > x) {
            h = x - x_curr;
        }

        k1 = h * dydx(x_curr, y);
```

```

        k2 = h * dydx(x_curr + h / 2, y + k1 / 2);

        y = y + k2;
        x_curr = x_curr + h;

        if (std::isinf(y) || std::isnan(y)) {
            std::cout << "Numerical instability detected at x = " <<
                x_curr << std::endl;
            break;
        }

        std::cout << "x = " << x_curr << ", y = " << y << std::endl;
    }
}

int main() {
    double x0 = 1.0;
    double y0 = 1.0;
    double x = 2.0;
    double h = 0.01;

    rungeKutta2(x0, y0, x, h);

    return 0;
}

```

```

x = 1.01, y = 1.00005
x = 1.02, y = 1.0002
x = 1.03, y = 1.00045
x = 1.04, y = 1.0008
x = 1.05, y = 1.00125
x = 1.06, y = 1.0018
x = 1.07, y = 1.00245
x = 1.08, y = 1.0032
x = 1.09, y = 1.00405
x = 1.1, y = 1.00501

```

Figure 2.5: RK2 Values

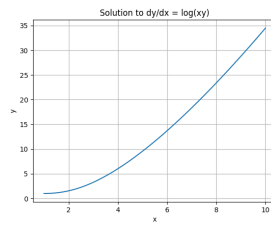


Figure 2.6: Original $dy/dx = \log(xy)$

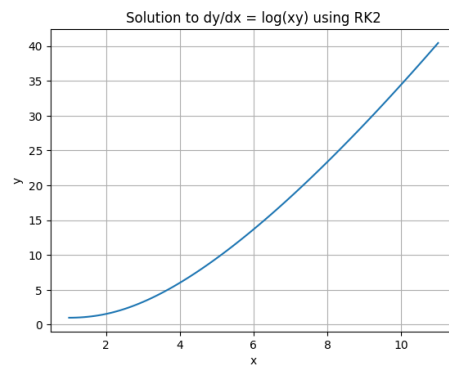


Figure 2.7: RK2 collection of points

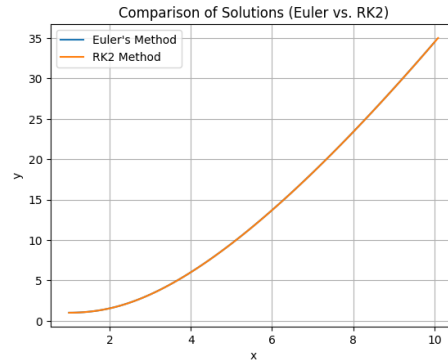


Figure 2.8: Compared Graph

2.6.2 RK3

$$\begin{aligned}
 k_1 &= \Delta t f(t_n, x_n) \\
 k_2 &= \Delta t f\left(x_0 + \frac{t}{2}, y_0 + \frac{k_1}{2}\right) \\
 k_3 &= \Delta t f\left(x + \Delta t, y + 2k_2 - k_1\right) \\
 y_1 &= y + \frac{1}{6}(k_1 + 4k_2 + k_3)
 \end{aligned} \tag{2.11}$$

2.6.3 RK4

The most widely known member of the Runge–Kutta family is generally referred to as "RK4", the "classic Runge–Kutta method".

$$\begin{aligned}
 k_1 &= \Delta t f(t_n, x_n) \\
 k_2 &= \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right) \\
 k_3 &= \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + k_2\right) \\
 k_4 &= \Delta t f\left(t_n + \Delta t, x_n + k_3\right) \\
 x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
 \end{aligned} \tag{2.12}$$

```

#include <iostream>
#include <cmath>

using namespace std;

// Function to compute the derivative f(x) = sin(|x-1|)
double f(double x) {
    return sin(abs(x-1));
}

```

```

}

// RK4 function to solve the differential equation
void rk4(double x0, double y0, double h, int n, double* x, double* y) {
    double k1[2], k2[2], k3[2], k4[2];

    for (int i=0; i<n; i++) {
        k1[0] = f(x0);
        k1[1] = y0;

        k2[0] = f(x0 + 0.5*h);
        k2[1] = y0 + 0.5*h*k1[0];

        k3[0] = f(x0 + 0.5*h);
        k3[1] = y0 + 0.5*h*k2[0];
        k4[0] = f(x0 + h);
        k4[1] = y0 + h*k3[0];

        x[i+1] = x0 + h;
        y[i+1] = y0 + (h/6.0)*(k1[1] + 2*k2[1] + 2*k3[1] + k4[1]); // y[i+1]
        // = y0 + (h/6.0)*(k1[1] + 2*k2[1] + 2*k3[1] + k4[1])
        x0 = x[i+1];
        y0 = y[i+1];
    }
}

int main() {
    double x0 = 0.0;
    double y0 = 1.0;
    double h = 0.01;
    int n = 1000;
    double* x = new double[n+1];
    double* y = new double[n+1];

    x[0] = x0;
    y[0] = y0;

    rk4(x0, y0, h, n, x, y);

    for (int i=0; i<=n; i++) {
        cout << x[i] << " " << y[i] << endl;
    }

    delete[] x;
    delete[] y;

    return 0;
}

```

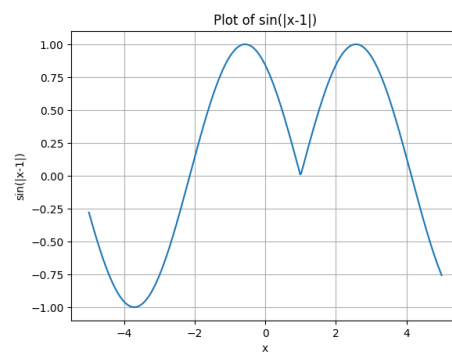


Figure 2.9: $f(x) = \sin(|x-1|)$

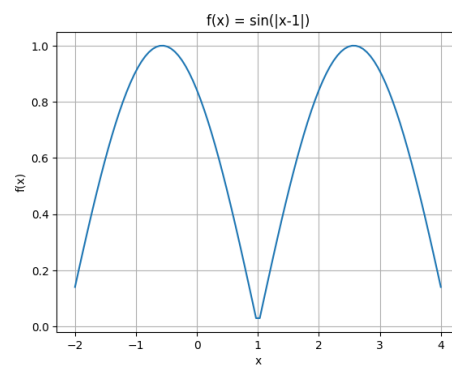


Figure 2.10: RK4

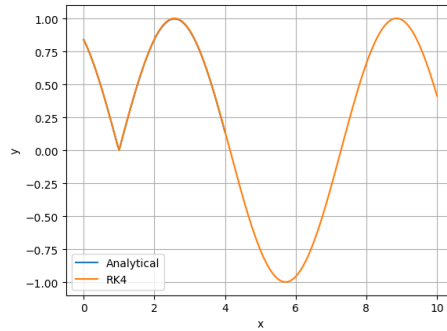


Figure 2.11: Compared Graph

2.7 Adam's Predictor and Corrector Methods

Adam's Predictor formula is:

$$y_{n+1,p} = y_n + \frac{h}{24}(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3}) \quad (2.13)$$

Adam's Corrector Formula is :

$$y_{n+1,p} = y_n + \frac{h}{24}(9y'_{n+1} - 19y'_n + 5y'_{n-1} - y'_{n-2}) \quad (2.14)$$

```
#include <iostream>
#include <cmath>

using namespace std;

double derivative(double x, double y) {
    return x + y/x - y;
}

void adam_bashforth(double& x, double& y, double h, int n) {
    double y_pred, y_corr;

    y_pred = y + h / 24 * (9 * derivative(x, y) + 19 * derivative(x + h,
        y) - 5 * derivative(x + h, y_pred) + derivative(x + 2 * h,
        y_pred));

    y_corr = y + h / 24 * (9 * derivative(x + h, y_corr) + 19 *
        derivative(x + h, y_pred) - 5 * derivative(x + h, y) +
        derivative(x + 2 * h, y_pred));

    x += h;
    y = y_corr;
}
```

```

x = 1.1, y = 1.10363
x = 1.2, y = 1.21087
x = 1.3, y = 1.32166
x = 1.4, y = 1.43592
x = 1.5, y = 1.55356
x = 1.6, y = 1.67447
x = 1.7, y = 1.79851
x = 1.8, y = 1.92557
x = 1.9, y = 2.05551
x = 2, y = 2.1882

```

Figure 2.12: Adam's Method Values

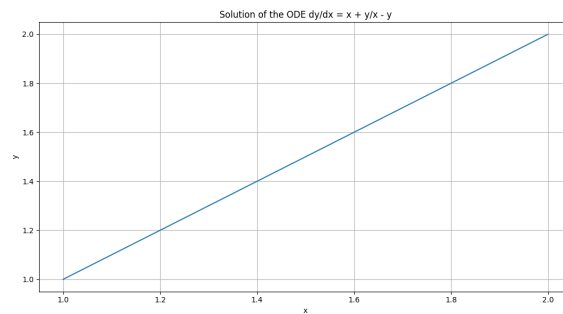


Figure 2.13: Curve of ODE $\frac{dy}{dx} = \frac{x+y}{x-y}$ (Matplotlib graph)

```

int main() {
    double x = 1.0, y = 1.0;
    double h = 0.1;
    int n = 10;

    for (int i = 0; i < n; i++) {
        adam_bashforth(x, y, h, n);
        cout << "x = " << x << ", y = " << y << endl;
    }

    return 0;
}

```

All the codes for graphs are available at my Github : <https://github.com/avroj1t/Numerical-Analysis>

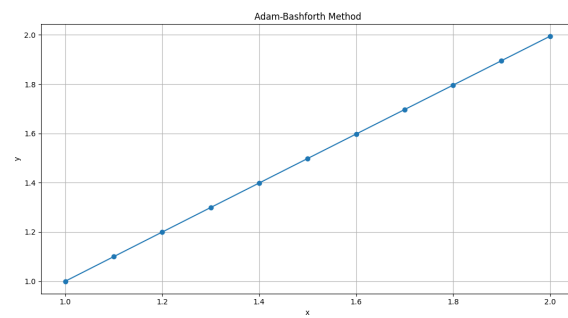


Figure 2.14: Adam's Method Solution of the expression

Chapter 3

Order of Convergence

The order of convergence refers to the rate at which an iterative sequence or method approaches its limit (or solution). In numerical analysis, it quantifies how quickly an approximation improves as the number of iterations increases. In simpler terms, the order of convergence tells you how quickly an approximation gets closer to the true solution as you keep repeating a method.

3.1 Theory

Let $\{\alpha\}_0^n$ be the sequence that converges to α and $e_n = \alpha_n - \alpha$

If there exists positive constants M and p ,

$$\frac{|\alpha_{n+1} - \alpha|}{|\alpha_n - \alpha|^p} = \frac{|e_{n+1}|}{|e_n|^p}$$
$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^p} = M$$

We say α_n converges to p .

3.1.1 Local Truncation Error

The Local Truncation Error (LTE) refers to the error introduced in a single step of the numerical method. If x_n is the approximation obtained at the n th step, and $x_{\text{exact}}(t_n)$ is the exact value at the same point, the LTE at step n is given by:

$$\text{LTE}_n = \frac{|x_{\text{exact}}(t_n) - \hat{x}_n|}{h}$$

where,

$x_{\text{exact}}(t_n)$ is the exact solution at the same step,

x_n is the approximation of the solution at step n (before the error accumulates

from subsequent steps

h is the step size (in methods like numerical integration, differential equations, etc.).

3.1.2 Global Error

The global error is the cumulative error at the final step after all iterations. If α_n is the approximate solution at the n_{th} step, and α^* is the exact solution, the global error E_n^* at the n_{th} step is defined as:

$$E_n = \alpha_n - \alpha^*$$

This measures the difference between the computed solution and the true solution after all iterations or steps have been completed.

Also,

$$GE = \sum_{n=1}^N LTE$$

AND

$$GE = \mathcal{O}(h^p)$$

where p is the order of convergence

3.2 Order of Convergence of Some Numerical Methods

3.2.1 Picard's Method

Given the initial value problem:

$$y'(t) = f(t, y), \quad y(t_0) = y_0$$

The solution to the ODE is approximated iteratively by Picard's method. The method starts with an initial approximation $y_0(t) = y_0$ and iteratively updates the solution using the following formula:

$$y_{n+1}(t) = y_0 + \int_{t_0}^t f(s, y_n(s)) ds$$

Where $y_n(t)$ is the approximation of the solution at the n -th iteration. The goal is to improve the accuracy of the approximation $y_n(t)$ as n increases.

To derive the order of convergence, let $y(t)$ denote the exact solution of the ODE, and $y_n(t)$ denote the approximation at the n -th step. The error at the n -th step is defined as:

$$E_n(t) = |y_n(t) - y(t)|$$

To find the error at the $(n + 1)$ -th iteration, substitute the expression for $y_{n+1}(t)$ into the error formula:

$$E_{n+1}(t) = |y_{n+1}(t) - y(t)| = \left| y_0 + \int_{t_0}^t f(s, y_n(s)) ds - y(t) \right|$$

Rewriting the integral expression:

$$E_{n+1}(t) = \left| \int_{t_0}^t (f(s, y_n(s)) - f(s, y(s))) ds \right|$$

Using the ****Lipschitz continuity**** of f with respect to y , we assume there exists a constant L such that:

$$|f(s, y_n(s)) - f(s, y(s))| \leq L|y_n(s) - y(s)|$$

This leads to the following error bound:

$$E_{n+1}(t) \leq L \int_{t_0}^t E_n(s) ds$$

Assuming that the error at each step behaves similarly, we approximate $E_n(s) \approx E_n(t_0)$, yielding:

$$E_{n+1}(t) \leq L E_n(t_0) \cdot (t - t_0)$$

Next, we analyze the ratio of errors at successive steps to determine the order of convergence. The ratio of the errors at steps $n + 1$ and n is given by:

$$\frac{E_{n+1}(t)}{E_n(t)} \leq L \cdot E_n(t_0)$$

This shows that the error decreases by a constant factor, and the convergence is **linear**. Therefore, the method has an Order of Convergence $p = 1$, meaning that the error decreases linearly with each iteration.

The Picard method converges linearly to the exact solution with order $p = 1$.

3.2.2 Runge Kutta Method Of 2nd Order

Let the exact solution be $t_{n+1} = t_n + h$ be denoted by $y(t_{n+1}) = y_n$ Using Taylor's Series,

$$y_{t_{n+1}} = y_{t_n} + y'(t_n)h + \frac{1}{2!}y''(t_n)h^2 + \frac{1}{3!}y'''(t_n)h^3 + \mathcal{O}(h^4) \quad (3.1)$$

Using the RK2 approximation,

$$y_{n+1} = y_n + f(t_n, y_n)h + \frac{1}{2!}f'(t_n, x_n)h^2 + \mathcal{O}(h^3) \quad (3.2)$$

Comparing both of them,

$$\text{Error} = y_{t_{n+1}} - y_{n+1}$$

$$\text{Error} = y_{t_{n+1}} - y_{n+1} = \left[y_{t_n} + y'(t_n)h + \frac{1}{2!}y''(t_n)h^2 + \frac{1}{3!}y'''(t_n)h^3 + \mathcal{O}(h^4) \right] - \left[y_n + f(t_n, y_n)h + \frac{1}{2!}f'(t_n, x_n)h^2 + \frac{1}{3!}f''(t_n, x_n)h^3 + \mathcal{O}(h^4) \right] \quad (3.3)$$

$$\text{Error} = \mathcal{O}(h^3)$$

Note : The dominant unmatched order determines the leading LTE unlike the Big O notation where we take the maximum between both of them. This is because we use Taylor's series approximation here.

Moving on, We earlier stated that

$$\begin{aligned} \text{GE} &= \sum_{n=1}^N \epsilon_n \\ &= N \mathbf{x} \epsilon_n \\ &= N \mathbf{x} \mathcal{O}(h^3) \end{aligned}$$

$$N = \frac{T - t_0}{h} = 1/h$$

Therefore,

$$GE = \mathcal{O}(h^2)$$

We also earlier stated that,

$$\text{GE} = \mathcal{O}(h^p)$$

$$\mathbf{p} = 2$$

Another Proof:

Let's take $\alpha_n = h/2$, $\alpha_{n+1} = h$, $\alpha_{n-1} = h/4$

$$\epsilon_{n+1} = |\alpha_{n+1} - \alpha_n|$$

$$\epsilon_n = |\alpha_n - \alpha_{n-1}|$$

We know that,

$$p = \log_q \left(\frac{\alpha_{n+1} - \alpha_n}{\alpha_n - \alpha_{n-1}} \right)$$

where,

$$q = \frac{h_{n+1}}{h_n}$$

$$q = 2$$

$$p = \log_2\left(\frac{\alpha_{n+1} - \alpha_n}{\alpha_n - \alpha_{n-1}}\right)$$

now,

$$\epsilon_{n+1} = \mathcal{O}(h^2/4)$$

$$\epsilon_n = \mathcal{O}(h^2/16)$$

Therefore, we can see

$$p = 2$$

3.2.3 Runge Kutta Method of Order 4

The Taylor expansion of $y(t_n + h)$ is given as:

$$y(t_n + h) = y(t_n) + hf(t_n, y_n) + \frac{h^2}{2}f'(t_n, y_n) + \frac{h^3}{6}f''(t_n, y_n) + \frac{h^4}{24}f^{(3)}(t_n, y_n) + O(h^5).$$

The 4th-order Runge-Kutta method is defined as:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

where:

$$\begin{aligned} k_1 &= hf(t_n, y_n), \\ k_2 &= hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right), \\ k_3 &= hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right), \\ k_4 &= hf(t_n + h, y_n + k_3) \end{aligned}$$

Let's take $\alpha_n = h/2$, $\alpha_{n+1} = h$, $\alpha_{n-1} = h/4$

We earlier stated that

$$LTE_{RK4} = \mathcal{O}(h^4)$$

So,

$$p = \log_q\left(\frac{\alpha_{n+1} - \alpha_n}{\alpha_n - \alpha_{n-1}}\right)$$

$$q = 4$$

We get,

$$p = \log_q\left(\frac{\alpha_{n+1} - \alpha_n}{\alpha_n - \alpha_{n-1}}\right)$$

But here we get,

$$\epsilon_{n+1} = \mathcal{O}(h^2/4)$$

$$\epsilon_n = \mathcal{O}(h^2/16)$$

We can see ,

$$p = 4$$

Chapter 4

Partial Differential Equations

The key defining property of a partial differential equation (PDE) is that there is more than one independent variable x, y, \dots . There is a dependent variable that is an unknown function of these variables $u(x, y, \dots)$. A PDE is an identity that relates the independent variables, the dependent variable u , and the partial derivatives of u .

$$F(x, y, u(x, y), u_x(x, y), u_y(x, y)) = F(x, y, u, u_x, u_y) = 0.$$

This is the most general PDE in two independent variables of first order. The order of an equation is the highest derivative that appears.

The most general second-order PDE in two independent variables is

$$F(x, y, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) = 0$$

Some examples of PDEs (all of which occur in physical theory) are:

1. $u_x + u_y = 0$ (transport)
2. $u_x + yu_y = 0$ (transport)
3. $u_x + u_{yy} = 0$ (shock wave)
4. $u_{xx} + u_{yy} = 0$ (Laplace's equation)

4.0.1 Linearity

Write the equation in the form $lu = 0$, where l is an operator. That is, if v is any function, lv is a new function. For example, $l = \frac{\partial}{\partial x}$ is the operator that takes v into its partial derivative v_x .

The definition of linearity is the following,

$$\mathcal{L}(u + v) = \mathcal{L}u + \mathcal{L}v$$

$$\mathcal{L}(cu) = c\mathcal{L}(u)$$

for any functions u, v and any constant c .

4.1 First Order Equations

A first order partial differential equation is a relation of the form.

$$F(x, y, u(x, y), u_x(x, y), u_y(x, y)) = F(x, y, u, u_x, u_y) = 0.$$

4.1.1 Solving Methods

Constant Coefficient Method

Consider the equation

$$au_x + bu_y = 0$$

where a and b are constants not both zero.

We can just use the separation of variables method.

Variable Coefficient Method

The equation

$$u_x + yu_y = 0$$

is linear and homogenous but has a variable coefficient.

Let us take an example where,

$$7u_x + 2xu_y = 0$$

which we can also write in the form

$$(7, 2x) \cdot \nabla u = 0$$

Now we can take it as a simple ode problem

$$\frac{dy}{dx} = \frac{2x}{7}$$

Integrating the both sides,

$$\int dy = \int \frac{2x dx}{7}$$

Therefore,

$$y = \frac{x^2}{2} = c$$

4.2 Types of First-Order Differential Equations

First-order differential equations involve a function and its first derivative. They can be classified into several types, each with unique characteristics and solution methods.

4.2.1 Separable Equations

- **Form:** $\frac{dy}{dx} = f(x)g(y)$
- **Example:** $\frac{dy}{dx} = x^2y$
- **Solution Method:** Separate the variables and integrate both sides.

4.2.2 Linear Equations

- **Form:** $\frac{dy}{dx} + p(x)y = q(x)$
- **Example:** $\frac{dy}{dx} + 2xy = x^3$
- **Solution Method:** Use an integrating factor to solve.

4.2.3 Exact Equations

- **Form:** $M(x, y)dx + N(x, y)dy = 0$, where $\frac{\partial M}{\partial y} = \frac{\partial N}{\partial x}$
- **Example:** $(2x + y)dx + (x - 2y)dy = 0$
- **Solution Method:** Integrate M with respect to x and N with respect to y , then equate the results to find the solution.

4.2.4 Homogeneous Equations

- **Form:** $\frac{dy}{dx} = f\left(\frac{y}{x}\right)$
- **Example:** $\frac{dy}{dx} = \frac{y^2 + x^2}{xy}$
- **Solution Method:** Use the substitution $v = \frac{y}{x}$ to transform the equation into a separable one.

4.2.5 Bernoulli Equations

- **Form:** $\frac{dy}{dx} + p(x)y = q(x)y^n$
- **Example:** $\frac{dy}{dx} + y = xy^2$
- **Solution Method:** Use the substitution $v = y^{1-n}$ to transform the equation into a linear one.

4.3 Second Order Equations

Second Order PDEs

A linear second order partial differential equation can be written as

$$A \frac{\partial^2 \phi}{\partial x^2} + B \frac{\partial^2 \phi}{\partial x \partial y} + C \frac{\partial^2 \phi}{\partial y^2} = F(x, y, \phi, \phi_x, \phi_y)$$

where A, B and C may be functions of x and y. Based on the local value of the coefficients, the equations are classified as follows:

$B^2 - 4AC > 0$	Hyperbolic
$B^2 - 4AC = 0$	Parabolic
$B^2 - 4AC < 0$	Elliptic

Note that an equation may change type from one point to another since the coefficients may be functions of x and y. We will typically assume that, when we say that an equation is of a given type, it remains of the same type over the whole domain.

Consider a valid change of independent variables $\xi = \xi(x, y)$, $\eta = \eta(x, y)$, such that

$$J = \begin{vmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{vmatrix} \neq 0.$$

Then,

$$\begin{aligned} \phi_x &= \phi_\xi \xi_x + \phi_\eta \eta_x \\ \phi_y &= \phi_\xi \xi_y + \phi_\eta \eta_y \\ \phi_{xx} &= \phi_{\xi\xi} \xi_x^2 + 2\phi_{\xi\eta} \xi_x \eta_x + \phi_{\eta\eta} \eta_x^2 + \phi_\xi \xi_{xx} + \phi_\eta \eta_{xx} \\ \phi_{yy} &= \phi_{\xi\xi} \xi_y^2 + 2\phi_{\xi\eta} \xi_y \eta_y + \phi_{\eta\eta} \eta_y^2 + \phi_\xi \xi_{yy} + \phi_\eta \eta_{yy} \\ \phi_{xy} &= \phi_{\xi\xi} \xi_x \xi_y + \phi_{\xi\eta} (\xi_x \eta_y + \xi_y \eta_x) + \phi_{\eta\eta} \eta_x \eta_y + \phi_\xi \xi_{xy} + \phi_\eta \eta_{xy} \end{aligned}$$

The transformed equation becomes

$$a\phi_{\xi\xi} + b\phi_{\xi\eta} + c\phi_{\eta\eta} = F(\xi, \eta, \phi, \phi_\xi, \phi_\eta)$$

with

$$\begin{aligned} a &= A\xi_x^2 + B\xi_x\xi_y + C\xi_y^2 \\ b &= 2A\xi_x\eta_x + B(\xi_x\eta_y + \xi_y\eta_x) + 2C\xi_y\eta_y \\ c &= A\eta_x^2 + B\eta_x\eta_y + C\eta_y^2 \end{aligned}$$

THEOREM: This classification is invariant under valid non-singular transformations.

Proof: From above $b^2 - 4ac = (B^2 - 4AC)(\xi_x\eta_y - \xi_y\eta_x)^2 = (B^2 - 4AC)|J|^2$

CANONICAL FORMS

HYPERBOLIC case ($B^2 - 4AC > 0$):

In this case it is always possible to choose ξ, η so that $a = c = 0$, i.e.

$$\begin{aligned} A \left(\frac{\zeta_x}{\zeta_y} \right)^2 + B \left(\frac{\zeta_x}{\zeta_y} \right) + C &= 0, \\ A \left(\frac{\eta_x}{\eta_y} \right)^2 + B \left(\frac{\eta_x}{\eta_y} \right) + C &= 0, \\ \zeta_x &= \frac{-B + \sqrt{B^2 - 4AC}}{2A} \zeta_y, \\ \eta_x &= \frac{-B - \sqrt{B^2 - 4AC}}{2A} \eta_y. \end{aligned}$$

Then, the equation becomes

$$\phi_{\zeta\eta} = F(\zeta, \eta, \phi, \phi_\zeta, \phi_\eta).$$

An alternative form can be obtained by setting $X = \zeta + \eta$, $Y = \zeta - \eta$:

$$\phi_{XX} - \phi_{YY} = F'(X, Y, \phi, \phi_X, \phi_Y).$$

PARABOLIC case ($B^2 - 4AC = 0$):

Here, we can only set a (or c) to zero (not both), otherwise ζ and η are not independent. If we set $a = 0$, then

$$\frac{\zeta_x}{\zeta_y} = -\frac{B}{2A}.$$

It can be verified, by direct evaluation, that in this case $b = 0$, in which case we can pick η to be any function such that $|J| \neq 0$, and the equation becomes:

$$\phi_{\eta\eta} = F(\zeta, \eta, \phi, \phi_\zeta, \phi_\eta).$$

ELLIPTIC case ($B^2 - 4AC < 0$):

This case is identical to the hyperbolic case but now ζ and η are complex conjugates ($B^2 - 4AC < 0$). Take $X = \zeta + \eta$, $Y = i(\zeta - \eta)$ and the equation becomes:

$$\phi_{XX} + \phi_{YY} = F'(X, Y, \phi, \phi_X, \phi_Y).$$

4.4 The Heat Equation

In this section we begin a study of the one-dimensional diffusion equation

$$u_t = ku_{xx} \quad (1)$$

Diffusions are very different from waves, and this is reflected in the mathematical properties of the equations. Because (1) is harder to solve than the wave equation, we begin this section with a general discussion of some of the properties of diffusions. We begin with the maximum principle, from which we'll deduce the uniqueness of an initial-boundary problem.

Maximum Principle. If $u(x, t)$ satisfies the diffusion equation in a rectangle (say, $0 \leq x \leq l$, $0 \leq t \leq T$) in space-time, then the maximum value of $u(x, t)$ is assumed either initially ($t = 0$) or on the lateral sides ($x = 0$ or $x = l$) (see Figure 1).

The minimum value has the same property; it too can be attained only on the bottom or the lateral sides. To prove the minimum principle, just apply the maximum principle to $[-u(x, t)]$

4.4.1 Uniqueness

Uniqueness is a solution dependent on the initial and boundary problems. The maximum principle can be used to give a proof of uniqueness for the Dirichlet problem for the diffusion equation. That is, there is at most one solution of

$$\begin{aligned} u_t - ku_{xx} &= f(x, t) \quad \text{for } 0 < x < l \text{ and } t > 0 \\ u(x, 0) &= \phi(x) \\ u(0, t) &= g(t) \\ u(l, t) &= h(t) \end{aligned} \quad (4.1)$$

for four given functions f , ϕ , g , and h . Uniqueness means that any solution is determined completely by its initial and boundary conditions. Indeed, let $u_1(x, t)$ and $u_2(x, t)$ be two solutions of (3). Let $w = u_1 - u_2$ be their difference. Then $w_t - kw_{xx} = 0$, $w(x, 0) = 0$, $w(0, t) = 0$, $w(l, t) = 0$. Let $T > 0$. By the maximum principle, $w(x, t)$ has its maximum for the rectangle on its

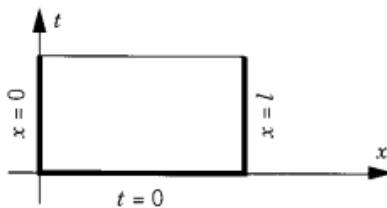


Figure 4.1: Figure 1

bottom or sides—exactly where it vanishes. So $w(x, t) \leq 0$. The same type of argument for the minimum shows that $w(x, t) \geq 0$. Therefore, $w(x, t) \equiv 0$, so that $u_1(x, t) \equiv u_2(x, t)$ for all $t \geq 0$.

4.4.2 Stability

This is the third ingredient of well-posedness (see Section 1.5). It means that the initial and boundary conditions are correctly formulated. The energy method leads to the following form of stability of problem (3), in case $h = g = f = 0$. Let $u_1(x, 0) = \phi_1(x)$ and $u_2(x, 0) = \phi_2(x)$. Then $w = u_1 - u_2$ is the solution with the initial datum $\phi_1 - \phi_2$. So from (4) we have

$$\int_0^l [u_1(x, t) - u_2(x, t)]^2 dx \leq \int_0^l [\phi_1(x) - \phi_2(x)]^2 dx. \quad (4.2)$$

It's just that nearness of the initial data points is always greater than or equal to the nearness of any two given solutions at any time.

4.4.3 Diffusion on the Whole Line

We consider the one-dimensional diffusion equation:

$$\frac{\partial u}{\partial t} = k \frac{\partial^2 u}{\partial x^2},$$

with the initial condition:

$$u(x, 0) = \phi(x),$$

where $-\infty < x < \infty$ and k is a positive constant.

Fundamental Solution

We begin by finding a particular solution $Q(x, t)$ that satisfies the diffusion equation and the initial condition $Q(x, 0) = \delta(x)$, where $\delta(x)$ is the Dirac delta function. This special solution is known as the **fundamental solution** or **Green's function**.

It can be shown that the fundamental solution is given by:

$$Q(x, t) = \frac{1}{\sqrt{4\pi kt}} \exp\left(-\frac{x^2}{4kt}\right).$$

General Solution

To obtain the general solution for any initial condition $\phi(x)$, we utilize the principle of superposition. We express $\phi(x)$ as a continuous sum of Dirac delta functions:

$$\phi(x) = \int_{-\infty}^{\infty} \phi(y) \delta(x-y) dy.$$

Based on the principle of superposition, the solution $u(x, t)$ can be expressed as:

$$u(x, t) = \int_{-\infty}^{\infty} \phi(y) Q(x-y, t) dy.$$

Substituting the expression for $Q(x, t)$, we arrive at the general solution:

$$u(x, t) = \frac{1}{\sqrt{4\pi kt}} \int_{-\infty}^{\infty} \phi(y) \exp\left(-\frac{(x-y)^2}{4kt}\right) dy.$$

4.5 Reflections of Waves

Now we try the same kind of problem for the wave equation as we did in Section 3.1 for the diffusion equation. We again begin with the Dirichlet problem on the half-line $(0, \infty)$. Thus the problem is:

$$\text{DE: } v_{tt} - c^2 v_{xx} = 0 \quad \text{for } 0 < x < \infty \text{ and } -\infty < t < \infty$$

$$\text{IC: } v(x, 0) = \phi(x), \quad v_t(x, 0) = \psi(x) \quad \text{for } t = 0 \text{ and } 0 < x < \infty$$

$$\text{BC: } v(0, t) = 0 \quad \text{for } x = 0 \text{ and } -\infty < t < \infty.$$

The reflection method is carried out in the same way as in Section 3.1. Consider the odd extensions of both of the initial functions to the whole line, $\phi_{odd}(x)$ and $\psi_{odd}(x)$. Let $u(x, t)$ be the solution of the initial-value problem on $(-\infty, \infty)$ with initial ϕ_{odd} and ψ_{odd} . Then $u(x, t)$ is once again an odd function of x . Therefore, $u(0, t) = 0$, so that the boundary condition is satisfied automatically. Define $v(x, t) = u(x, t)$ for $0 < x < \infty$ [the restriction of u to the half-line]. Then $v(x, t)$ is precisely the solution we are looking for. For $x \geq 0$,

$$v(x, t) = u(x, t) = \frac{1}{2} [\phi_{odd}(x+ct) + \phi_{odd}(x-ct)] + \frac{1}{2c} \int_{x-ct}^{x+ct} \psi_{odd}(y) dy.$$

Let's "unwind" this formula, recalling the meaning of the odd extensions. First we notice that for $x > c|t|$, only positive arguments occur in the formula, so that $u(x, t)$ is given by the usual formula:

$$v(x, t) = \frac{1}{2} [\phi(x+ct) + \phi(x-ct)] + \frac{1}{2c} \int_{x-ct}^{x+ct} \psi(y) dy \quad (2) \quad (4.3)$$

for $x > c|t|$.

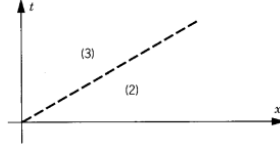


Figure 1

Figure 4.2:

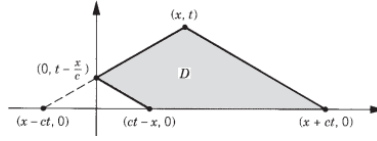


Figure 2

Figure 4.3: Enter Caption

But in the other region $0 < x < c|t|$, we have $\phi_{odd}(x - ct) = -\phi(ct - x)$, and so on, so that

$$v(x, t) = \frac{1}{2} [\phi(x + ct) - \phi(ct - x)] + \frac{1}{2c} \int_0^{x+ct} \psi(y) dy + \frac{1}{2c} \int_{x-ct}^0 [-\psi(-y)] dy.$$

Notice the switch in signs! In the last term we change variables $y \rightarrow -y$ to get $\frac{1}{2c} \int_{ct-x}^{ct+x} \psi(y) dy$. Therefore,

$$v(x, t) = \frac{1}{2} [\phi(ct + x) - \phi(ct - x)] + \frac{1}{2c} \int_{ct-x}^{ct+x} \psi(y) dy \quad (3) \quad (4.4)$$

for $0 < x < c|t|$. The complete solution is given by the pair of formulas (2) and (3). The two regions are sketched in Figure 1 for $t > 0$.

Chapter 5

Boundary Value Problems

5.1 The Dirichlet Condition

$$u_{tt} = c^2 u_{xx} \quad \text{for } 0 < x < l \quad (1)$$

$$u(0, t) = 0 = u(l, t) \quad (2)$$

$$u(x, 0) = \phi(x), \quad u_t(x, 0) = \psi(x). \quad (3)$$

with some initial conditions

The method we shall use consists of building up the general solution as a linear combination of special ones that are easy to find.

A separated solution is a solution of (1) and (2) of the form

$$u(x, t) = X(x)T(t). \quad (4)$$

(It is important to distinguish between the independent variable written as a lowercase letter and the function written as a capital letter.)

Plugging the form (4) into the wave equation (1), we get

$$X(x)T''(t) = c^2 X''(x)T(t)$$

or, dividing by $-c^2 XT$,

$$-\frac{T''}{c^2 T} = -\frac{X''}{X} = \lambda.$$

This defines a quantity λ , which must be a constant. (Proof: $d\lambda/dx = 0$ and $d\lambda/dt = 0$, so λ is a constant. Alternatively, we can argue that λ doesn't depend on x because of the first expression, and doesn't depend on t because of the second expression, so that it doesn't depend on any variable.) We will show at the end of this section that $\lambda > 0$. (This is the reason for introducing the minus signs the way we did.)

So let $\lambda = \beta^2$ where $\beta > 0$. Then the equations above are a pair of separate (!) ordinary differential equations for $X(x)$ and $T(t)$:

$$X'' + \beta^2 X = 0 \quad \text{and} \quad T'' + c^2 \beta^2 T = 0. \quad (5)$$

These ODEs are easy to solve. The solutions have the form

$$X(x) = C \cos \beta x + D \sin \beta x \quad (6)$$

$$T(t) = A \cos \beta ct + B \sin \beta ct. \quad (7)$$

where A, B, C, and D are constants.

The second step is to impose the boundary conditions (2) on the separated solution. They simply require that $X(0) = 0 = X(l)$. Thus $0 = X(0) = C$ and $0 = X(l) = D \sin(\beta l)$.

Surely we are not interested in the obvious solution $C = D = 0$. So we must have $\beta l = n\pi$, a root of the sine function. That is,

$$\lambda_n = \left(\frac{n\pi}{l}\right)^2, \quad X_n(x) = \sin\left(\frac{n\pi x}{l}\right) \quad (n = 1, 2, 3, \dots) \quad (8)$$

are distinct solutions. Each sine function may be multiplied by an arbitrary constant.

Therefore, there are an infinite (!) number of separated solutions of (1) and (2), one for each n. They are

$$u_n(x, t) = \left(A_n \cos\left(\frac{n\pi ct}{l}\right) + B_n \sin\left(\frac{n\pi ct}{l}\right) \right) \sin\left(\frac{n\pi x}{l}\right)$$

($n = 1, 2, 3, \dots$), where A_n and B_n are arbitrary constants. The sum of solutions is again a solution, so any finite sum

$$u(x, t) = \sum_{n=1}^{\infty} \left(A_n \cos\left(\frac{n\pi ct}{l}\right) + B_n \sin\left(\frac{n\pi ct}{l}\right) \right) \sin\left(\frac{n\pi x}{l}\right) \quad (9)$$

Formula (9) solves (3) as well as (1) and (2), provided that

$$\phi(x) = \sum_n A_n \sin \frac{n\pi x}{l} \quad (10)$$

and

$$\psi(x) = \sum_n \frac{n\pi c}{l} B_n \sin \frac{n\pi x}{l}. \quad (11)$$

5.2 The Neumann Boundary

The same method works for both the Neumann and Robin boundary conditions (BCs). In the former case, (4.1.2) is replaced by $u_x(0, t) = u_x(l, t) = 0$. Then the eigenfunctions are the solutions $X(x)$ of

$$\begin{aligned} -X'' &= \lambda X, \\ X'(0) &= X'(l) = 0, \end{aligned} \tag{1}$$

other than the trivial solution $X(x) \equiv 0$.

As before, let's first search for the positive eigenvalues $\lambda = \beta^2 > 0$. As in (4.1.6), $X(x) = C \cos \beta x + D \sin \beta x$, so that

$$X'(x) = -C\beta \sin \beta x + D\beta \cos \beta x.$$

The boundary conditions (1) mean first that $0 = X'(0) = D\beta$, so that $D = 0$, and second that

$$0 = X'(l) = -C\beta \sin \beta l.$$

Since we don't want $C = 0$, we must have $\sin \beta l = 0$. Thus $\beta = \pi/l, 2\pi/l, 3\pi/l, \dots$. Therefore, we have the

<p>Eigenvalues: $\left(\frac{\pi}{l}\right)^2, \left(\frac{2\pi}{l}\right)^2, \dots$</p> <p>Eigenfunctions: $X_n(x) = \cos \frac{n\pi x}{l} \quad (n = 1, 2, \dots)$</p>
--

Next let's check whether zero is an eigenvalue. Set $\lambda = 0$ in the ODE (1). Then $X'' = 0$, so that $X(x) = C + Dx$ and $X'(x) = D$. The Neumann boundary conditions are both satisfied if $D = 0$. C can be any number. Therefore, $\lambda = 0$ is an eigenvalue, and any constant function is its eigenfunction.

If $\lambda < 0$ or if λ is complex (nonreal), it can be shown directly, as in the Dirichlet case, that there is no eigenfunction. Therefore, the list of all the eigenvalues is

$$\lambda_n = \left(\frac{n\pi}{l}\right)^2 \quad \text{for } n = 0, 1, 2, 3, \dots \tag{4}$$

5.3 The Robin Condition

We continue the method of separation of variables for the case of the Robin condition. The Robin condition means that we are solving $-X'' = \lambda X$ with the boundary conditions

$$\begin{aligned} X' - a_0 X &= 0 & \text{at } x = 0 \\ X' + a_l X &= 0 & \text{at } x = l. \end{aligned} \tag{2}$$

The two constants a_0 and a_l should be considered as given.

The physical reason they are written with opposite signs is that they correspond to radiation of energy if a_0 and a_l are positive, absorption of energy if a_0 and a_l are negative, and insulation if $a_0 = a_l = 0$. This is the interpretation for heat problem: See the discussion in Section 1.4 or Exercise 2.3.8. For the case of the vibrating string, the interpretation is that the string shares its energy with the endpoints if a_0 and a_l are positive, whereas the string gains some energy from the endpoints if a_0 and a_l are negative: See Exercise 11.

The mathematical reason for writing the constants in this way is that the unit outward normal \mathbf{n} for the interval $0 \leq x \leq l$ points to the left at $x = 0$ ($n = -1$) and to the right at $x = l$ ($n = +1$). Therefore, we expect that the nature of the eigenfunctions might depend on the signs of the two constants in opposite ways.

Chapter 6

Fourier Series

Fourier Series

Let $f(x)$ be defined in the interval $(-L, L)$ and outside of this interval by $f(x + 2L) = f(x)$, i.e., $f(x)$ is $2L$ -periodic. It is through this avenue that a new function on an infinite set of real numbers is created from the image on $(-L, L)$. The *Fourier series* or *Fourier expansion* corresponding to $f(x)$ is given by

$$\frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right) \quad (6.1)$$

where the *Fourier coefficients* a_n and b_n are

$$\begin{aligned} a_n &= \frac{1}{L} \int_{-L}^L f(x) \cos \frac{n\pi x}{L} dx \\ b_n &= \frac{1}{L} \int_{-L}^L f(x) \sin \frac{n\pi x}{L} dx \quad n = 0, 1, 2, \dots \end{aligned} \quad (6.2)$$

ORTHOGONALITY CONDITIONS FOR THE SINE AND COSINE FUNCTIONS

Notice that the Fourier coefficients are integrals. These are obtained by starting with the series, (1), and employing the following properties called orthogonality conditions:

- (a) $\int_{-L}^L \cos \frac{m\pi x}{L} \cos \frac{n\pi x}{L} dx = 0$ if $m \neq n$ and L if $m = n$
- (b) $\int_{-L}^L \sin \frac{m\pi x}{L} \sin \frac{n\pi x}{L} dx = 0$ if $m \neq n$ and L if $m = n$ (3)
- (c) $\int_{-L}^L \sin \frac{m\pi x}{L} \cos \frac{n\pi x}{L} dx = 0$. Where m and n can assume any positive integer values.

6.1 The Complex Form

Begin with a real periodic function $F(t)$ with period T :

$$F(t+T) = F(t). \quad (6.3)$$

We are going to write this as a series in complex exponentials

$$F(t) = a_0 + a_1 e^{i\omega t} + a_2 e^{2i\omega t} + \cdots + a_n e^{ni\omega t} + \cdots \\ + a_{-1} e^{-i\omega t} + a_{-2} e^{-2i\omega t} + \cdots + a_{-n} e^{-ni\omega t} + \cdots$$

using both positive and negative indices, so we can write

$$F(t) = \sum_{n=-\infty}^{+\infty} a_n e^{in\omega t}. \quad (6.4)$$

Here $\omega = 2\pi/T$. Note that since we have assumed $F(t)$ is real, the complex coefficients on the right side above must add up to a real result. Now we will integrate over a period $T = 2\pi/\omega$:

$$\int_0^{2\pi/\omega} F(t) dt = \int_0^{2\pi/\omega} \left(\sum_{n=-\infty}^{+\infty} a_n e^{in\omega t} \right) dt \quad (6.5)$$

and assuming we can interchange integration and summation:

$$= \sum_{n=-\infty}^{+\infty} a_n \int_0^{2\pi/\omega} e^{in\omega t} dt \quad (6.6)$$

Now consider the integral in the right-hand side, for various values of n .

$$\int_0^{2\pi/\omega} e^{in\omega t} dt = \frac{1}{i\omega n} (e^{2\pi i n} - 1) = 0 \text{ for } n \neq 0, \text{ and} \\ = \int_0^{2\pi/\omega} dt = 2\pi/\omega = T \text{ for } n = 0.$$

This gives us

$$\int_0^T F(t) dt = a_0 T, \quad (6.7)$$

$$a_0 = \frac{1}{T} \int_0^T F(t) dt. \quad (6.8)$$

which is the average value of F .

6.2 Back to the real form

To return to nonnegative indices and real sines and cosines is just a matter of rewriting now. Write the sum in eq. 3 in separate parts:

$$F(t) = \sum_{n=-\infty}^{-1} a_n e^{in\omega t} + a_0 + \sum_{n=1}^{+\infty} a_n e^{in\omega t} \quad (6.9)$$

$$= \sum_{n=1}^{\infty} a_{-n} e^{-in\omega t} + a_0 + \sum_{n=1}^{+\infty} a_n e^{in\omega t} \quad (6.10)$$

$$= a_0 + \sum_{n=1}^{\infty} (a_n e^{in\omega t} + a_{-n} e^{-in\omega t}) \quad (6.11)$$

Using Euler's formula $e^{i\phi} = \cos \phi + i \sin \phi$,

$$F(t) = a_0 + \sum_{n=1}^{\infty} (a_n + a_{-n}) \cos n\omega t + \sum_{n=1}^{\infty} i(a_n - a_{-n}) \sin n\omega t. \quad (6.12)$$

We now let

$$A_n = a_n + a_{-n}, \quad B_n = i(a_n - a_{-n}), \quad \frac{A_0}{2} = a_0, \quad (6.13)$$

and get

$$F(t) = \frac{A_0}{2} + \sum_{n=1}^{\infty} A_n \cos n\omega t + \sum_{n=1}^{\infty} B_n \sin n\omega t, \quad (6.14)$$

the usual real form of the Fourier series. We can work out the coefficient explicitly to see where all the constants come from:

$$A_n = a_n + a_{-n} = \frac{1}{T} \int_0^T F(t) (e^{-in\omega t} + e^{in\omega t}) \quad (6.15)$$

$$= \frac{2}{T} \int_0^T F(t) \cos n\omega t, \quad (6.16)$$

and

$$B_n = i(a_n - a_{-n}) = \frac{1}{T} \int_0^T F(t) i(e^{-in\omega t} - e^{in\omega t}) \quad (6.17)$$

$$= \frac{2}{T} \int_0^T F(t) \sin n\omega t. \quad (6.18)$$

6.3 Magnitude-angle form

Yet another form of the Fourier series can be obtained from eq. 22 by writing

$$A_n \cos n\omega t + B_n \sin n\omega t = C_n \cos(n\omega t - \phi_n) \quad (6.19)$$

$$= C_n \cos n\omega t \cos \phi_n + C_n \sin n\omega t \sin \phi_n. \quad (6.20)$$

Equate the coefficients of like terms

$$A_n = C_n \cos \phi_n, \quad B_n = C_n \sin \phi_n, \quad (6.21)$$

and so

$$C_n = \sqrt{A_n^2 + B_n^2}, \quad \phi_n = \arctan \frac{B_n}{A_n}. \quad (6.22)$$

The magnitude-angle form of the series is thus

$$F(t) = \frac{A_0}{2} + \sum_{n=1}^{\infty} C_n \cos(n\omega t - \phi_n) \quad (6.23)$$

or

$$F(t) = \frac{A_0}{2} + \sum_{n=1}^{\infty} C_n \sin(n\omega t + \frac{\pi}{2} - \phi_n). \quad (6.24)$$

Chapter 7

Finite Element Methods

7.1 Introduction

The finite element method (FEM) is a numerical technique for solving problems which are described by partial differential equations or can be formulated as functional minimization. A domain of interest is represented as an assembly of *finite elements*. Approximating functions in finite elements are determined in terms of nodal values of a physical field which is sought. A continuous physical problem is transformed into a discretized finite element problem with unknown nodal values. For a linear problem a system of linear algebraic equations should be solved. Values inside finite elements can be recovered using nodal values.

Two features of the FEM are worth to be mentioned:

1. Piece-wise approximation of physical fields on finite elements provides good precision even with simple approximating functions (increasing the number of elements we can achieve any accuracy).
2. Locality of approximation leads to sparse equation systems for a discretized problem. This helps to solve problems with very large number of nodal unknowns.

7.2 The Galerkin Method

The Galerkin Method introduces the concept of Weighted Residuals and Weak Formulation of Differential Equations, what it essentially does is that it takes a differential equation, commonly in a weak formulation and converts it to a discrete problem by applying linear constraints determined by finite sets of shape functions, to approximate the solution of the equation. Before that, going through some definitions are important.

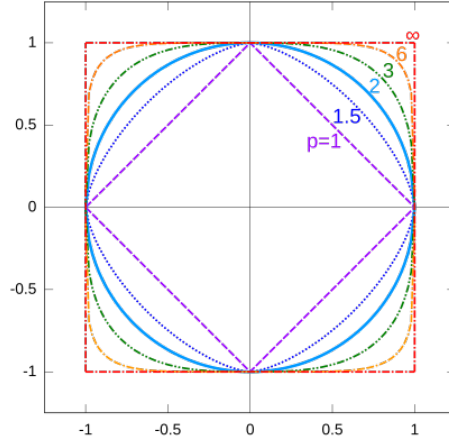


Figure 7.1: An Illustration of Unit Circles of L-p Spaces

7.2.1 L-p Spaces

L-p Spaces is a way to measure the "size" or "magnitude" of a vector or a function in a finite dimensional vector space. These are spaces of functions whose p-th power of the absolute value is integrable. These are spaces when imagined in the nth dimensional level can be said to lie in between the Taxicab Norm and Maximum Norm.

Taxicab Norm : $\|x\| = \sum x_i$

Maximum Norm : $\|x\|_\infty = \max(\|x\|_1, \|x\|_2, \dots, \|x\|_n)$

L_1 space is the Taxicab Norm.

L_2 space is the Euclidean Norm.

L_∞ space is the Maximum Norm.

A more generalised form to write L-p spaces would be:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{1/p}$$

for $p \geq 1$

7.2.2 Sobolev Spaces

Sobolev space is a vector space of functions equipped with a norm that is a combination of Lp-norms of the function together with its derivatives up to a given order.

$$\Omega \subset \mathbb{R}^n$$

$$\Omega = \text{Boundary of } \Omega$$

Let $m > 1$ be an integer. Let $1 < p < \infty$. The Sobolev Space $W^{m,p}(\Omega)$ is defined by

$$W^{m,p}(\Omega) = \left\{ u \in L^p(\Omega) \mid \mathcal{D}^\alpha u \in L^p(\Omega), \forall \alpha \in \mathbb{N}^n \right\}$$

This is a vector subspace of $L^p(\Omega)$ and can be embedded into the field norm.

$$1 \leq p < \infty$$

$$|u|_{m,p,\Omega} = \sum_{|\alpha| \leq m} \|\mathbb{D}^\alpha u\|_{\mathcal{L}^p(\Omega)}^{(1/p)}$$

The concept of "derivatives" in Sobolev spaces is often generalized to "weak derivatives," which allows us to work with functions that may not have classical derivatives in the usual sense. This is particularly important when dealing with solutions to partial differential equations (PDEs), which may not always be smooth. In simpler terms, a Sobolev space provides a framework for dealing with functions that have "enough" derivatives to be useful in various applications, especially in the study of PDEs.

Sobolev spaces generalize the notion of differentiability to functions with "weak" (distributional) derivatives. For a domain $\Omega \subset \mathbb{R}^n$:

- $L^2(\Omega)$: Space of square-integrable functions.
- $H^1(\Omega)$: Functions $u \in L^2(\Omega)$ with weak first derivatives $\partial_i u \in L^2(\Omega)$.
- $H_0^1(\Omega)$: Subspace of $H^1(\Omega)$ where functions vanish on the boundary $\partial\Omega$ (for Dirichlet BCs).

These spaces are Hilbert spaces equipped with norms, e.g.,

$$\|u\|_{H^1(\Omega)} = \left(\int_{\Omega} |u|^2 + |\nabla u|^2 dx \right)^{1/2}.$$

7.2.3 Weak Formulation of Sobolev Spaces

Consider the **Poisson Equation** where

$$-\Delta v = f \text{ in } \Omega, u = 0 \text{ on } \partial\Omega$$

1. **Multiply by a test function:** Choose $v \in \mathbb{H}_0^1(\Omega)$, multiply with both sides, and integrate.

$$-\int_{\Omega} (\Delta u) v dx = \int_{\Omega} f v dx$$

2. **Integrate by parts:** Using Green's Identity we get,

$$\int_{\Omega} \nabla u \cdot \nabla v dx - \int_{\Omega} (\nabla u \cdot n) v ds = \int_{\Omega} f v dx$$

3. **Enforce Boundary Conditions:** Boundary conditions vanish as $v=0$

$$\int_{\Omega} \nabla u \cdot \nabla v dx = \int_{\Omega} f v dx$$

The weak problem is stated as:

$$\text{Find } u \in H_0^1(\Omega) \text{ such that } a(u, v) = L(v) \quad \forall v \in H_0^1(\Omega),$$

where:

- **Bilinear form:** $a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v dx,$
- **Linear form:** $L(v) = \int_{\Omega} f v dx.$

7.2.4 Coming back to Galerkin Method

(a) Problem Setting

Consider a boundary value problem (BVP) defined by:

$$L(u) = f \quad \text{in } \Omega, \quad \text{with BCs on } \partial\Omega,$$

where L is a differential operator (linear or nonlinear), u is the unknown solution, and f is a forcing term.

(b) Weak (Variational) Formulation

1. **Multiply by a test function** $v \in \tilde{V}$, where \tilde{V} is a test space.
2. **Integrate over the domain** Ω :

$$\int_{\Omega} L(u) v d\Omega = \int_{\Omega} f v d\Omega \quad \forall v \in \tilde{V}.$$

3. **Integrate by parts** to reduce derivative orders, leading to:

$$a(u, v) = L(v),$$

where $a(\cdot, \cdot)$ is a **bilinear form** and $L(\cdot)$ is a **linear form**.

The Galerkin method approximates the solution u in a **finite-dimensional subspace** $V_h \subset V$:

$$u_h(x) = \sum_{i=1}^N c_i \phi_i(x),$$

where $\{\phi_i\}_{i=1}^N$ are **basis functions** (e.g., piecewise polynomials in FEM).

Key Principle: Orthogonality of the Residual

The residual $R = L(u_h) - f$ is forced to be orthogonal to V_h :

$$\int_{\Omega} R\phi_j d\Omega = 0 \quad \forall j = 1, \dots, N.$$

This generates a system of equations:

$$\sum_{i=1}^N c_i a(\phi_i, \phi_j) = \langle f, \phi_j \rangle \quad \forall j,$$

leading to the linear system:

$$Ac = \mathbf{b},$$

where $A_{ij} = a(\phi_i, \phi_j)$ (stiffness matrix) and $b_j = \langle f, \phi_j \rangle$ (load vector).

7.3 Shape Functions

Shape functions interpolate nodal values to approximate the solution $u_h(x)$ within each element. For example:

$$u_h(x) = \sum_{i=1}^N N_i(x)u_i,$$

where $N_i(x)$ are shape functions and u_i are nodal coefficients.

• Some Properties Regarding Shape Functions

1. Kronecker Delta Property:

$$N_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Ensures $u_h(x_i) = u_i$, directly enforcing nodal values.

2. Partition of Unity:

$$\sum_{i=1}^N N_i(x) = 1 \quad \forall x \in \text{element.}$$

Guarantees a constant field is represented exactly.

3. Continuity:

- \mathbb{C}_1 (e.g., Hermite polynomials): Derivatives are continuous (used for beam bending)
- \mathbb{C}_0 (e.g., linear elements): Functions are continuous across element boundaries.

7.4 Solving 1D Poisson Equation using the Galerkin Method in C++ and Gnuplot

1DGalerkin.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>

// 1D Poisson Equation Using Galerkin Method
//STEPS
//1. Discretize the domain into N elements
//2. Define the basis functions (linear in this case)
//3. Assemble the stiffness matrix and load vector
//4. Apply boundary conditions
//5. Solve the linear system using Gaussian elimination
//6. Output the solution to a file
//7. Visualize the solution using gnuplot or any other tool

double f(double x) {
    return 1.0; // constant source
}

int main() {
    int N = 10; // Number of elements
    double L = 1.0;
    double h = L / N;
    std::vector<double> x(N + 1);
    for (int i = 0; i <= N; ++i)
        x[i] = i * h;

    // Initialize stiffness matrix A and load vector b
    std::vector<std::vector<double>> A(N + 1, std::vector<double>(N + 1,
        0.0));
    std::vector<double> b(N + 1, 0.0);

    // Assembly loop
    for (int i = 1; i < N; ++i) {
        A[i][i - 1] += -1.0 / h;
        A[i][i] += 2.0 / h;
        A[i][i + 1] += -1.0 / h;

        b[i] += h * f(x[i]);
    }

    // Boundary conditions u(0)=u(1)=0
    A[0][0] = A[N][N] = 1.0;
    b[0] = b[N] = 0.0;
```

```

// Solve linear system (simple Gaussian elimination)
std::vector<double> u(N + 1, 0.0);
for (int i = 1; i < N; ++i) {
    double pivot = A[i][i];
    for (int j = i + 1; j <= N; ++j)
        A[i][j] /= pivot;
    b[i] /= pivot;

    for (int k = i + 1; k < N; ++k) {
        double factor = A[k][i];
        for (int j = i; j <= N; ++j)
            A[k][j] -= factor * A[i][j];
        b[k] -= factor * b[i];
    }
}

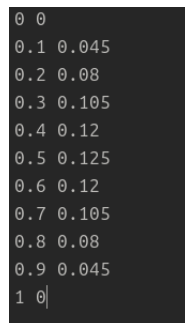
// Back substitution
for (int i = N - 1; i >= 1; --i) {
    u[i] = b[i];
    for (int j = i + 1; j < N; ++j)
        u[i] -= A[i][j] * u[j];
}

// Output to file
std::ofstream fout("solution1D.txt");
for (int i = 0; i <= N; ++i)
    fout << x[i] << " " << u[i] << "\n";

fout.close();
std::cout << "1D solution written to solution1D.txt\n";
return 0;
}

```

which gives us the Values :



```

0 0
0.1 0.045
0.2 0.08
0.3 0.105
0.4 0.12
0.5 0.125
0.6 0.12
0.7 0.105
0.8 0.08
0.9 0.045
1 0

```

Figure 7.2: Values

Plotting it on GnuPlot:
plot1D.gp

```
set terminal pngcairo size 800,600 enhanced font 'Arial,12'
set output 'solution1D.png'

set title '1D Poisson Equation Solution'
set xlabel 'x'
set ylabel 'u(x)'
set grid

plot 'solution1D.txt' using 1:2 with linespoints lt rgb "blue" lw 2 pt 7
title 'u(x)'
```

we get this graph:

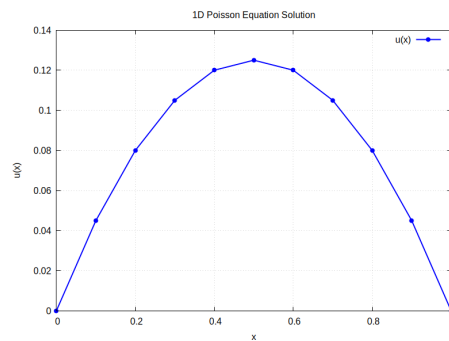


Figure 7.3: Graph of 1D Poisson

7.5 MESH

In the Finite Element Method (FEM), a mesh is a discretization of a complex geometric domain into smaller, simpler shapes called elements. These elements are interconnected at points called nodes, forming a grid-like structure. The mesh serves as the foundation for approximating the solution to partial differential equations (PDEs) over the domain.

Components of a mesh:

1. Nodes:

- Discrete points where the solution (e.g., displacement, temperature) is calculated.
- Nodes define the vertices/corners of elements.
- Example: In 1D, nodes are points along a line; in 2D/3D, they are coordinates in space.

2. Elements:

- Simple geometric shapes (e.g., lines, triangles, quadrilaterals, tetrahedrons) that connect nodes.
- The solution is approximated within each element using **shape functions**.

3. Edges/Faces:

- Boundaries of elements (e.g., edges of triangles, faces of tetrahedrons).

7.5.1 Types of Meshes

Table 7.1: Types of Meshes

Category	Description	Example Elements
1D	Line segments	Linear elements (2-node)
2D	Triangles, quadrilaterals	Tri3 (triangle), Quad4 (quadrilateral)
3D	Tetrahedrons, hexahedrons, prisms	Tet4 (tetrahedron), Hex8 (hexahedron)
Structured	Uniform, grid-like arrangement (easy to generate but less flexible)	Cartesian grids
Unstructured	Irregular arrangement (flexible for complex geometries)	Triangulated surfaces
Adaptive	Refined locally in regions of interest (e.g., high gradients or errors)	Hybrid meshes
Conforming	Elements align perfectly at shared boundaries	Standard FEM meshes
Non-Conforming	Elements may not align (used in discontinuous Galerkin methods)	Hanging nodes allowed

7.5.2 Simple Mesh Implementation using Python and Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
import math

#0 - Define Parameters
L = 0.1
```

```

h = 0.05
r = 0.02

nb_element = 20

#1-1 Create Nodes
#Nodes = np.array([[0,0],[0,1],[1,0],[1,1]])

Nodes = []

for x in np.linspace(0, L, num=nb_element):
    if(x<r):
        y0 = math.sqrt(r**2-x**2)
        for y in np.linspace(y0, h, num=nb_element):
            Nodes.append([x,y])

    else:
        for y in np.linspace(0, h, num=nb_element):
            Nodes.append([x,y])

#1-2 Display Nodes

points = np.array(Nodes)

plt.plot(points[:,0],points[:,1], 'o')
plt.show()

#2-1 Create Elements
from scipy.spatial import Delaunay
tri = Delaunay(points)

#2-2 Display Elements
plt.triplot(points[:,0], points[:,1], tri.simplices)
plt.plot(points[:,0], points[:,1], 'o')
plt.show()

#2-3 Cleanup Mesh
#Create a set of points on a circle of diameter 0.0195
p = []
r2=0.0195
for x in np.linspace(0,r2,10):
    p.append([x,math.sqrt(r2**2-x**2)])

#Find the elements which contain those points
tri.find_simplex(p)

#Feed the result of the previous function to the np.delete method
#Create a new set of elements without the problematic elements
mesh = np.delete(tri.simplices,[0,153,154,21],0)

```

```

#3- Export into a file
nb_nodes = len(points)
nb_elements = len(mesh)

file = open("plate_mesh.dat", "w")
file.write("{} {}\n".format(nb_nodes, nb_elements))
for i, node in enumerate(Nodes):
    file.write("{} {} {}\n".format(i, node[0], node[1]))
for j, elem in enumerate(mesh):
    file.write("{} {} {} {}\n".format(j, elem[0], elem[1], elem[2]))
file.close()

```

which gives us:

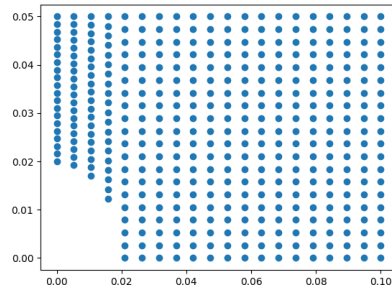


Figure 7.4: Result

Chapter 8

References

1. B.S. Grewal, *Numerical Methods in Engineering & Science with Programs in C, C++ & MATLAB*, Perfect Paperback.
2. Jeffrey Chasnov, *Numerical Methods for Engineers*.
3. Walter A. Strauss, *Partial Differential Equations: An Introduction*.
4. Peter J. Olver, *Introduction to Partial Differential Equations*.
5. Richard Haberman, *Partial Differential Equations*.
6. H. J. P. Arnoldi, *Fourier Series and Integrals*.
7. David W. Kammler, *Introduction to Fourier Analysis*.
8. G. B. Mathews, *Fourier Series*.
9. James Ward Brown and Ruel V. Churchill, *Fourier Series and Boundary Value Problems*.
10. R. D. N. S. Yada, *Fourier Series: An Introduction*.
11. J. N. Reddy, *An Introduction to the Finite Element Method*, McGraw-Hill Education.
12. Daryl L. Logan, *A First Course in the Finite Element Method*, Cengage Learning.
13. Olek C. Zienkiewicz and Robert L. Taylor, *The Finite Element Method: Its Basis and Fundamentals*, Elsevier.
14. S. S. Bhavikatti, *Finite Element Analysis*, New Age International.
15. J. David Logan, *Applied Partial Differential Equations*, available at: <https://math.unl.edu/jlogan1/chapters.pdf>

16. William E. Boyce and Richard C. DiPrima, *Elementary Differential Equations and Boundary Value Problems* (PDE section), PDF retrieved from: <https://people.math.sc.edu/BoyceDiPrima.pdf>
17. C. Pozrikidis, *Introduction to Partial Differential Equations*, Open Textbook Library, <https://open.umn.edu/opentextbooks/textbooks/introduction-to-pdes>
18. MIT OpenCourseWare, *18.152 Introduction to Partial Differential Equations*, Lecture Notes, <https://ocw.mit.edu/courses/mathematics/18-152-introduction-to-partial-differential-equations-fall-2011/lecture-notes/>