Finite element solution of the Reaction-Diffusion equation

by

Richard Kagisho Mahlakwana

DISSERTATION

submitted in fulfilment of the

requirements for the degree of

Master of Science

In

Applied Mathematics

In the

FACULTY OF SCIENCE AND AGRICULTURE

(**School of Mathematical and Computer Sciences**)

at the

UNIVERSITY OF LIMPOPO

Supervisor**:** Dr G T Marewo

2020

Declaration

I declare that the dissertation hereby submitted to the University of Limpopo, for the Master of Science in Applied Mathematics has not previously been submitted by me for a degree at this or any other university; that it is my work in design and in execution, and that all material contained herein has been duly acknowledged.

Mr Mahlakwana RK November 2020

Contents

Abstract

In this study we present the numerical solution of boundary value problems for the reaction-diffusion equations in 1-d and 2-d that model phenomena such as kinetics and population dynamics. These differential equations are solved numerically using the finite element method (FEM). The FEM was chosen due to several desirable properties it possesses and the many advantages it has over other numerical methods. Some of its advantages include its ability to handle complex geometries very well and that it is built on well established Mathematical theory, and that this method solves a wider class of problems than most numerical methods. The Lax-Milgram lemma will be used to prove the existence and uniqueness of the finite element solutions. These solutions are compared with the exact solutions, whenever they exist, in order to examine the accuracy of this method. The adaptive finite element method will be used as a tool for validating the accuracy of the FEM. The convergence of the FEM will be proven only on the real line.

Chapter 1

Introduction

The reaction-diffusion equations (RDEs) have been investigated by many researchers due to their many industrial applications. Such applications include describing pattern-formation phenomena in a variety of biological, chemical and physical system. RDEs can be used to describe the change in space and time of the concentration of one or more chemical substances. The spread of a population of many individuals in space may be modelled using RDEs. The transport processes and kinetics of substrate and redox-mediator in surface deposited film, or kinetics of electroenzymes at electrode/solution interface can be modelled by a reaction-diffusion equation.

RDEs have been at the centre of attention for many researchers over the years. For example, Alvarez-Ramirez [1] used a non-standard finite difference schemes to find a numerical solution for a non-linear reaction-diffusion equation in cylindrical and spherical coordinates. Their numerical results illustrated that the finite difference scheme based on Green's function formulations gives better results for a coarse-grained mesh compared to non-standard finite difference schemes. Ashyraliyev [3] used a finite volume method to find an approximate solution of 1-D and 2-D RDEs with singular reaction source terms. The challenge they had was that the solution they obtained was continuous, but not continuously differentiable. This was conquered by examining discretization on a number of special locally refined grids analytically in 1-D and experimentally in 2-D. Hence, the maximum norm second-order convergence was restored. A new Homotopy perturbation method was used by Shanthi and Rajendran [15] to find the approximate analytic solution to a non-linear reaction-diffusion equation with Michaelis-Menten kinetics. This method was discovered to be easy to apply.

Bhrawy and Doha [4] studied a non-linear time-dependent RDEs subject to Dirichlet boundary conditions using the spectral Jacobi-Gauss-Lobatto collocation method (SJ-G-LCM). The method made use of Jacobi polynomials and Gauss-Lobatto collocation points to construct semi-analytic solutions. They reduced the differential problem to systems of ordinary differential equations in the time variable and the systems were solved using standard numerical methods like Jacobi spectral collocation method. The SJ-G-LCM was found to be efficient and it produced accurate results with less than 10 collocation points. Wang and Zhang [17] solved systems of 2-dimensional non-linear time-dependent RDEs using a compact finite difference method (FDM). The systems were reduced to non-linear discrete systems which were solved using three monotone iterative algorithms. The compact FDM was discovered to be fourth order accurate in both space and time variables. In [9] Madzvamuse and Chung sought numerical solutions for systems of non-linear time dependent RDEs. In this study, the finite element method was used to discretize the problem in space. To discretize the problem in time the backward Euler method, Crank-Nicolson method and the fractional-step θ method were employed with a uniform time-step. The Newton method and the Picard iteration were used to handle the non-linearities of the systems. Their results showed that the fractional-step θ method coupled with a single Newton iteration at each time-step was as accurate as the fully adaptive Newton method, and that the Newton method outperformed the Picard iteration.

Non-linear partial differential equations (NPDEs) are not generally easy to solve and in most cases exact solutions may not exist. It is for these reasons that approximate analytic methods and numerical methods are used to approximate the solutions of NPDEs.

There are a number of approximate analytic methods to find solutions to various NPDEs. For example, Akram and Sadaf [2] used the homotopy analysis method (HAM) to solve a boundary value problem for a 9th order differential equation. They showed that the HAM is highly accurate. One feature of the HAM is that solutions are obtained in terms of infinite series. If the auxiliary parameter, auxiliary function, auxiliary linear operator and initial guess are not carefully chosen, then the infinite series may not converge. Kumar and Singh [7] used the homotopy analysis transform method (HATM) to determine the approximate solution of the Klein-Gordon equations arising in quantum field theory. They showed that the HATM is very efficient and gives highly accurate results. Another approximate analytic method is the variational iteration method (VIM). It was applied to different NPDEs by various researchers [16, 6] and it was discovered that the VIM takes a few iterations to obtain highly accurate solutions for the problems they studied.

Lots of numerical methods have also been used for approximating NPDEs. Makanda [10] used the spectral quasilinearization method, spectral relaxation method and spectral local linearization method to solve various systems of differential equations modelling fluid flow problems. The results indicated that these methods are efficient, they produce accurate results with only few iterations, and they posses straightforward algorithms. The finite difference method (FDM) [14, 12] was used to solve nonlinear differential equations such as the Fokker-Planck equation and the Boltzmann transport equation. The FDM was shown to give highly accurate numerical solutions. However, it is generally difficult to implement the FDM on irregular geometries or with complex boundary conditions. A numerical method which handles irregular geometries much better is the FEM. Among the many authors who used this method, Chaudhari and Pate [5] used it to solve Poisson's equation. The FEM is an efficient discretization technique for solving differential equations accurately using a considerably small number of grid points. The theory behind the FEM provides error estimates which can be used to establish convergence of numerical solutions to the exact solution. However, the FEM is computationally intensive. With the advent of fast computers recently, this disadvantage is overshadowed by merits of the method.

In this study, boundary value problems for reaction-diffusion equations in 1-d and 2-d are numerically solved using the finite element method. This numerical method was chosen due to its many desirable properties, and it also has many advantages over other numerical methods. For example, the finite element method handles complex geometries very well, its analysis makes use of well established branches of mathematics like functional analysis, and the method solves a wider class of problems than most numerical methods.

At this point we give an overview of the subsequent chapters. In Chapter 2, we make use of continuous piecewise polynomials to approximate continuous functions on the real line and in the real plane. We also derive error estimates for these approximations and we use these estimates to prove quadratic convergence. In Chapter 3, we introduce the finite element method for solving a boundary value problem that models the reaction-diffusion equation in the real line. This boundary value problem is converted to its equivalent variational formulation (weak formulation), then we seek approximate solution for the weak formulation from the space of all continuous piecewise linear functions. we prove the existence and uniqueness of the weak form solution, then prove some basic error estimates and use these estimates to formulate the adaptive finite element method. In chapter 4, we briefly demonstrate how a variational formulation is derived from its equivalent reaction-diffusion equation in the real plane. The existence and uniqueness of the solution for the variational formulation will be proved using the Lax-Milgram Lemma. The MATLAB's pdetool will be employed to find the approximate solution for the boundary value problem, and an illustrative example is given to investigate the computational efficiency of the method. Chapter 5 gives a conclusion to the study. This is followed by an appendix with MATLAB code for an example problem on 2-d. Finally, an appendix is given for the same problem that consists of code in Python.

Chapter 2

Approximating continuous functions

2.1 Introduction

In this chapter we introduce continuous piecewise polynomial functions which are used to approximate continuous functions on the real line and in the real plane. Error estimates and convergence for these approximations will be proved.

2.2 Piecewise polynomial approximation on the real line

Let $I = [x_0, x_1]$. We denote the space of all continuous functions on I_1 by $C(I)$, and the space of all linear functions on I_i by $P_1(I)$. The linear interpolant $\pi f \in P_1(I)$ of $f \in C(I)$ is given by

$$\pi f(x) = f(x_0)L_0(x) + f(x_1)L_1(x)$$

where

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \text{ and } L_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

Since

$$L_i(x_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad i, j = 0, 1.$$

then it follows that

$\pi f(x_0) = f(x_0)$ and $\pi f(x_1) = f(x_1)$.

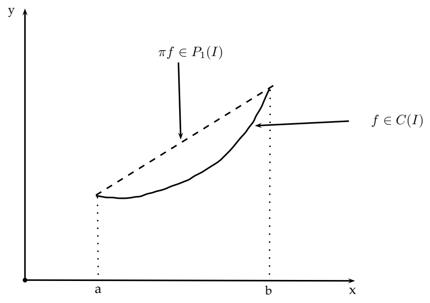


Figure 2.1: Linear interpolation on $I = [a, b] = [x_0, x_1]$.

Function f is interpolated by πf as shown in Figure 2.1. In the following theorem we estimate the error in approximating f with $\pi f(x_0)$ on $I = [x_0, x_1]$.

Theorem 1 (Error estimates for linear interpolant) *Linear interpolant πf satisfies the following estimates*

$$\|\pi f - f\|_{L^2(I)} \leq Ch^2\|f''\|_{L^2(I)} \text{ and} \tag{2.1}$$

$$\|(\pi f - f)'\|_{L^2(I)} \leq Ch\|f''\|_{L^2(I)} \tag{2.2}$$

where C is a constant, $\|f\|_{L^2(I)} := \int_I f(x)dx$ and $h = x_1 - x_0$.

Proof 1 *To prove inequality (2.2) we first let $e = \pi f - f$. We notice that $f \in C(I)$, $x_0 < x_1$, and $e(x_0) = 0$ but $\underline{e'(x_0) \neq 0}$. Rolle's theorem says that there exists $\bar{x} \in (x_0, x_1)$ such that $e'(\bar{x}) = 0$. Now, for any point $y \in (x_0, x_1)$ such that $y > \bar{x}$ we have*

$$\int_{\bar{x}}^{y} e''dx = e'(y) - e'(\bar{x}) = e'(y) \qquad (e'(\bar{x}) = 0) \tag{2.3}$$

$$\begin{aligned}
\Rightarrow e'(y) &\leq \int_{\bar{x}}^{y} |e''|dx \qquad \left(|\int f| \leq \int |f|\right) \qquad (2.4)\\
&\leq \int_{I_1} 1.|e''|dx \qquad ((\bar{x}, y) \subset (x_0, x_1) =: I_1)\\
&\leq \left(\int_{I_1} 1^2 dx\right)^{\frac{1}{2}} \left(\int_{I_1} |e''^2|dx\right)^{\frac{1}{2}} \qquad (\textit{Cauchy-Schwarz inequality})\\
&= h^{\frac{1}{2}}\|e''\|_{L^2(I_1)} \qquad \left(\|e''\|_{L^2(I_1)} = \left(\int_{I_1} |e''^2|dx\right)^{\frac{1}{2}}\right)
\end{aligned}$$

Thus, we have

$$e'(y)^2 \leq h\|e''\|^2_{L^2(I_1)} \tag{2.5}$$

which results from squaring both sides. Integrating inequality (2.5) with respect to y over I_1

$$\int_{I_1} e'(y)^2 \leq h\|e''\|^2_{L^2(I_1)} \int_{I_1} dy \tag{2.6}$$

$$\Rightarrow \|e'\|^2_{L^2(I_1)} \leq h^2 \|e''\|^2_{L^2(I_1)} \tag{2.7}$$

Taking square roots gives inequality (2.2).

Now, we prove inequality (2.1). Since $e(x_0) = \pi f(x_0) - f(x_0) = 0$, *for any point* y *in* I *we have*

$$\int_{x_0}^{y} e'(x)dx = e(y) - e(x_0) = e(y) \qquad (e(x_0) = 0) \tag{2.8}$$

An argument similar to that consisting of inequalities (2.4) through (2.7) gives

$$\|e\|_{L^2(I)} \leq h\|e'\|_{L^2(I)} \tag{2.9}$$

Therefore

$$\|e\|_{L^2(I)} \leq h\|e'\|_{L^2(I)} \leq h^2\|e''\|_{L^2(I)} = h^2\|f''\|_{L^2(I)}. \tag{2.10}$$

Let $I := [a, b]$, let $I = \cup_{i=1}^{n} I_i$ where $I_i := [x_{i-1}, x_i]$, and $a = x_0 < x_1 < \cdots < x_n = b$. Let V_h denote the space

$$V_h = \{v \in C(I) : v|_{I_i} \in P_1(I_i), I_i = [x_{i-1}, x_i], 1 \leq i \leq 1\}$$

of all continuous piecewise linear functions on I. The function $f \in C(I)$ is interpolated on I by $\pi f \in V_h$ given by

$$\pi f = \sum_{i=1}^{n} f(x_i)\varphi_i(x) \tag{2.11}$$

where

$$\varphi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h_i}, & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{h_{i+1}}, & x \in [x_i, x_{i+1}] \\ 0, & \text{otherwise} \end{cases}$$

is a hat function defined on I. Function f is interpolated by πf as shown in Figure 2.2. In the following theorem we estimate the error in approximation $\pi f \approx f$ on $I = [a, b] = \cup_{i=1}^{n} I_i$.

Theorem 2 *Linear interpolant* πf *satisfies the error estimate.*

$$\|\pi f - f\|_{L^2(I)} \leq Ch^2\|f\|_{L^2(I)} \tag{2.12}$$

where $h = \max_{1 \leq i \leq n} h_i$ *on* $I = [a, b]$.

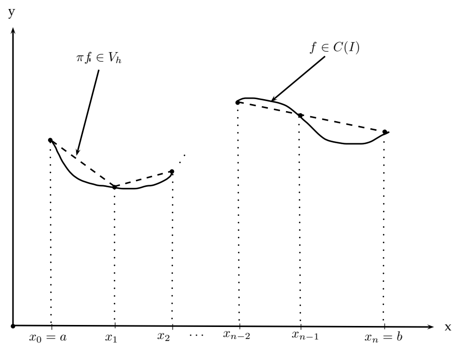


Figure 2.2: Linear interpolation on $I = [a, b]$.

Proof 2

$$\begin{aligned}
\|f - \pi f\|^2_{L^2(I)} &= \int_I (f - \pi f)^2 dx \qquad (2.13)\\
&= \sum_{i=1}^{n} \int_{I_i} (f - \pi f)^2\\
&\le \sum_{i=1}^{n} \left(Ch_i^2 \|f\|_{L^2(I_i)}\right)^2 \quad \left(\textit{since } \|f - \pi f\|_{L^2(I_i)} \le Ch_i^2 \|f\|_{L^2(I_i)}\right).\\
&\le C \sum_{i=1}^{n} h_i^4 \|f\|^2_{L^2(I_i)}\\
&\le C \sum_{i=1}^{n} h^4 \|f\|^2_{L^2(I_i)} \quad (h = \max_{1 \le i \le n} h_i).
\end{aligned}$$

(2.14)

Therefore

$$\|\pi f - f\|^2_{L^2(I)} \le Ch^4 \|f\|^2_{L^2(I)}. \qquad (2.15)$$

Now, taking the square roots of (2.15), we get

$$\|\pi f - f\|_{L^2(I)} \le Ch^2 \|f\|_{L^2(I)}. \qquad (2.16)$$

Corollary 1 *$\pi f \to f$ as $h \to 0$.*

Proof 3 *Since*

$$0 \le \|\pi f - f\|_{L^2(I)} \le h^2 \|f''\|_{L^2(I)} \to 0 \qquad (2.17)$$

as $h \to 0$, then it follows from squeeze theorem that

$$\pi f \to f \textit{ as } h \to 0.$$

2.3 Piecewise polynomial approximation in the real plane

Let $\Omega \in \mathbb{R}^2$ be a bounded domain with boundary $\partial\Omega$ in the real plane. We define a triangulation $\mathcal{K}$ of Ω as a set $\{K\}$ of triangles K such that $\Omega = \cup_{k\in\mathcal{K}} K$ and the intersection of two triangles is either an edge, a corner or empty. We refer to the corners of the triangles as nodes. Let h_k be the local mesh size (length of the longest edge in K), then $h = \max_{k\in\mathcal{K}} h_K$ is called a global mesh size of K.

A triangulation $\mathcal{K}$ with n_p nodes and n_t elements is stored in a computer as two matrices, P called the point matrix and T called the connectivity matrix. The matrix P is of order $2 \times n_p$ and T is of order $3 \times n_t$.

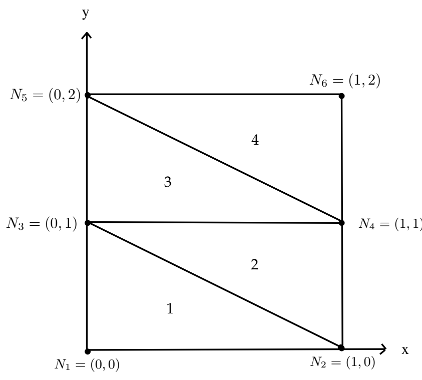


Figure 2.3: A simple mesh of a rectangular region.

Figure 2.3 shows a simple mesh on a rectangular region consisting of 6 nodes and 4 triangles. From Figure 2.3, we obtain the point matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{bmatrix}$$

and the connectivity matrix

$$T = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 4 & 6 \\ 3 & 3 & 5 & 5 \end{bmatrix}$$

The entries of the point matrix P in the j^{th} column are given by the coordinates $(x^{(j)}, y^{(j)})$ of the node N_j. The entries of the connectivity matrix T in the j^{th} column contains node numbers i, j and k for the nodes N_i, N_j and N_k of triangle K_j.

Let
$$P_1(K) = \{v : v = c_0 + c_1x_1 + c_2x_2, (x_1, x_2) \in K, c_0, c_1, c_2 \in \mathbb{R}\}$$

be the space of linear functions on triangle K. Given $f \in C(K)$, then on K :

$$f \approx \pi f := \sum_{j=1}^{N} f(N_i)\varphi_j(x,y)$$

where $\varphi_j \in P_1(K)$ and

$$\varphi_j(N_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j. \end{cases}$$

Define

$$Df := \sqrt{\left|\frac{\partial f}{\partial x}\right|^2 + \left|\frac{\partial f}{\partial y}\right|^2},$$

$$D^2 f := \sqrt{\left|\frac{\partial^2 f}{\partial x^2}\right|^2 + 2\left|\frac{\partial^2 f}{\partial y \partial x}\right|^2 + \left|\frac{\partial^2 f}{\partial y^2}\right|^2}$$

and let

$$\|f\|_{L^2(K)} := \left(\int_K f^2\right)^{\frac{1}{2}}.$$

Proposition 1 *Linear interpolant πf satisfies the error estimate.*

$$\|f - \pi f\|_{L^2(K)} \leq C h_K^2 \|D^2 f\|_{L^2(K)} \tag{2.18}$$

where h_K is the length of the longest edge in K, and C a constant. Proof was left out.

Define space

$$V_h = \{v : v \in C(\Omega), v|_K \in P_1(K), \forall K \in \mathcal{K}\}$$

of all continuous piecewise linear polynomials on Ω, where $C(\Omega)$ is the space of all continuous functions on Ω.

Given $f \in (\Omega)$, then on Ω

$$f \approx \pi f = \sum_{j=1}^{n_p} f(N_j)\varphi_j(x,y) \tag{2.19}$$

where $\{\varphi_j(x,y)\}_{j=1}^{n_p}$ is a basis (of hat functions) for the space V_h of all continous piecewise linear functions on K.

Proposition 2 *Linear interpolant πf satisfies the error estimate*

$$\|f - \pi f\|_{L^2(\Omega)}^2 \leq C \sum_{K \in \mathcal{K}} h_K^4 \|D^2 f\|_{L^2(\Omega)}^2 \tag{2.20}$$

where h_K is the length of the longest edge in K, and C a constant.

Proof 4

$$\begin{aligned}\|f-\pi f\|^2_{L^2(\Omega)} &= \int_\Omega (f-\pi f)^2 \qquad \left(\textit{since } \|f\|^2_{L^2(\Omega)} = \int_\Omega f^2\right)\\ &= \sum_{K\in\mathcal{K}} \int_K (f-\pi f)^2\\ &\leq \sum_{K\in\mathcal{K}} Ch_K^4 \|D^2 f\|^2_{L^2(K)} \qquad (\textit{by estimate } 2.18).\end{aligned}$$

Corollary 2 *$\pi f \to f$ as $h \to 0$ where $h = \max_{K\in\mathcal{K}} h_k$.*

Proof 5 *Since*

$$\|f-\pi f\|^2_{L^2(\Omega)} \leq \sum_{K\in\mathcal{K}} Ch_K^4 \|D^2 f\|^2_{L^2(K)} = Ch^4\|f''\|^2_{L^2(I)}. \tag{2.21}$$

Taking square roots in (2.21) we obtain that

$$0 \leq \|\pi f - f\|_{L^2(I)} \leq Ch^2\|f''\|_{L^2(I)} \to 0 \tag{2.22}$$

as $h \to 0$. Therefore, it follows from the squeeze theorem that

$$\|\pi f - f\|_{L^2(I)} \to 0 \text{ as } h \to 0. \tag{2.23}$$

Hence

$$\pi f \to f \text{ as } h \to 0$$

with rate $O(h^2)$.

2.4 Summary

In this chapter we approximated continuous functions on $\mathbb{R}$ using their continuous piecewise linear interpolants. We did the same thing in $\mathbb{R}^2$ and we derived estimates for these approximations. We also managed to use these error estimates to establish convergence for these approximations.

Chapter 3

Reaction-diffusion equation on the real line

3.1 Introduction

In this chapter the FEM is introduced is used to find a numerical solution of a boundary value problem for the RDE on the real line. Firstly, the boundary value problem is converted to its equivalent variational formulation. We shall make use of Lax-milgram lemma to prove that the variational formulation has a unique solution for two cases. The FEM will be used to seek the approximate solution in the space of continuous piecewise linear functions. This reduces the boundary value problem to a system of linear algebraic equations. The FEM will be implemented on a computer using Octave [13]. We derive an error estimate and use it to formulate an adaptive FEM. We also investigate whether or not the adaptive FEM gives more accurate results than the standard FEM. We will only do two illustrative examples.

3.2 Model problem

We consider the problem of finding u satifying the differential equation

$$-(pu')' + cu = f, x \in I = [0, L] \tag{3.1}$$

subject to boundary conditions

$$\begin{aligned} p(0)u'(0) &= c_0(u(0) - d_0) - e_0 && (3.2) \\ -p(L)u'(L) &= c_L(u(L) - d_L) - e_L && (3.3) \end{aligned}$$

where $p(x) > 0$, $c(x) \geqslant 0$ and $f(x)$ are given functions with $c_0 \geqslant 0, d_0, e_0, c_L \geqslant 0, d_L$ and e_L as given constants.

3.3 Variational formulation

We will transform equation (3.1) to its equivalent variational or weak form. First, we define the space of all functions that are square integrable on the interval $I := [0, L]$ and whose first derivatives are also square integrable on I, by:

$$V = \{v(x) : \|v\|_{L^2(I)} < \infty, \|v'\|_{L^2(I)} < \infty\}$$

where

$$\|v\|_{L^2(I)} = \left(\int_I |v|^2 dx\right)^{\frac{1}{2}}$$

is the L^2-norm of v on I. Now, multiplying equation (3.1) by a test function $v \in V$, and integrating by parts gives

$$-pu'v|_0^L + \int_0^L pu'v'dx + \int_0^L cuvdx = \int_0^L fvdx \tag{3.4}$$

A test function is a function that it:

1. is smooth (all its derivatives are continuous).
2. has compact support (i.e *supp* f is closed and bounded where *supp* $f := \{x : f(x) \neq 0\}$,).

Substituting boundary conditions (3.2) and (3.3) into (3.4), we get

$$(c_L(u(L) - d_L) - e_L)v(L) + (c_0(u(0) - d_0) - e_0)v(0) + \int_0^L (pu'v' + cuv) \\ = \int_0^L fv \tag{3.5}$$

Now, taking all the terms without the function u to the right hand side, we get the desired variational formulation: Find $u \in V$ satifying

$$c_L u(L)v(L) + c_0 u(0)v(0) + \int_0^L (pu'v' + cuv) \\ = \int_0^L fv + (c_L d_L + e_L)v(L) + (c_0 d_0 + e_0)v(0) \tag{3.6}$$

for all $v \in V$. We will show in section (3.5) that variational formulation (3.6) has a unique solution.

3.4 Finite element approximation

Let $0 = x_0 < x_1 < x_2 < \cdots < x_N = L$ be a partition of interval I. Associated with each node x_i is the hat function φ_i which was introduced in Section 2.2 We replace V by V_h so that the finite element method seeks $u_h \in V_h$ satifying

$$\begin{aligned} c_L u_h(L)v(L) + c_0 u_h(0)v(0) &+ \int_0^L (pu_h'v' + cu_hv) \\ &= \int_0^L fv + (c_Ld_L + e_L)v(L) + (c_0d_0 + e_0)v(0) \end{aligned} \tag{3.7}$$

for all $v \in V_h$. Replacing the function v with hat functions φ_i, $i = 0, 1, 2, ..., N$, changes equation (3.7) to

$$\begin{aligned} c_L u_h(L)\varphi_i(L) + c_0 u_h(0)\varphi_i(0) &+ \int_0^L pu_h'\varphi_i' + \int_0^L cu_h\varphi_i \\ &= \int_0^L f\varphi_i + (c_Ld_L + e_L)\varphi_i(L) + (c_0d_0 + e_0)\varphi_i(0) \end{aligned} \tag{3.8}$$

for $i = 0, 1, 2, ..., N$. Since $u_h \in V_h$, then

$$u_h = \sum_{j=0}^{N} \xi_j \varphi_j \tag{3.9}$$

where ξ_j, with $j = 0, 1, ..., N$ are unknown coefficients to be found. Consequently, equation (3.8) becomes

$$\begin{aligned} c_L \sum_{j=0}^{N} \xi_j\varphi_j(L)\varphi_i(L) + c_0 \sum_{j=0}^{N} \xi_j\varphi_j(0)\varphi_i(0) + \int_0^L p \sum_{j=0}^{N} \xi_j\varphi_j'\varphi_i' + \int_0^L c \sum_{j=0}^{N} \xi_j\varphi_j\varphi_i \\ = \int_0^L f\varphi_i + (c_Ld_L + e_L)\varphi_i(L) + (c_0d_0 + e_0)\varphi_i(0) \end{aligned}$$

with $i = 0, 1, ..., N$. Interchanging integration and summation, we get

$$\begin{aligned} c_L \sum_{j=0}^{N} \xi_j\varphi_j(L)\varphi_i(L) + c_0 \sum_{j=0}^{N} \xi_j\varphi_j(0)\varphi_i(0) + \sum_{j=0}^{N} \xi_j \int_0^L p\varphi_j'\varphi_i' + \sum_{j=0}^{N} \xi_j \int_0^L c\varphi_j\varphi_i \\ = \int_0^L f\varphi_i + (c_Ld_L + e_L)\varphi_i(L) + (c_0d_0 + e_0)\varphi_i(0) \end{aligned}$$

with $i = 0, 1, ..., N$, or in matrix form

$$(A + M + R)\xi = b + r \tag{3.10}$$

where

$$A_{ij} = \int_0^L p\varphi_j'\varphi_i', \tag{3.11}$$

$$M_{ij} = \int_0^L c\varphi_j\varphi_i, \tag{3.12}$$

$$R_{ij} = c_L\varphi_j(L)\varphi_i(L) + c_0\varphi_j(0)\varphi_i(0), \tag{3.13}$$

$$b_i = \int_0^L f\varphi_i \tag{3.14}$$

and

$$r_i = (c_Ld_L + e_L)\varphi_i(L) + (c_0d_0 + e_0)\varphi_i(0) \tag{3.15}$$

The matrix A is known as the stiffness matrix, M is the mass matrix, b is the load vector and R and r are boundary matrices. Upon solving linear system (3.10), coefficients $\xi_1, \xi_2, \cdots, \xi_{N-1}$ become known. Hence, we compute the desired solution using equation (3.9). Algorithm 1 shows the basic steps involved when implementing the previously discusssed finite element method.

Algorithm 1: A basic finite element method

```
1 Divide the interval $I$ into $N$ subintervals and define the corresponding
    space $V_h$ of continuous piecewise linear functions.
2 Compute the $(N+1)\times(N+1)$ matrices $A, M, R$ and $(N+1)\times 1$ matrices
    $b$ and $r$, with entries (3.11), (3.12), (3.13), (3.14) and (3.15) respectively
3 Solve the linear system $(A+M+R)\xi = b+r$.
4 Set $u_h = \sum_{j=0}^{N} \xi_j\varphi_j$.
```

3.5 Existence and uniqueness of weak solution

3.5.1 A simple problem

We consider a simple case of (3.1) by setting $c = 0$, so that equation (3.1) simplifies to

$$-(pu')' = f, x \in I \tag{3.16}$$

Taking $c_0 \longrightarrow \infty$ and setting $d_0 = 0$ reduces the boundary condition (3.2) to the Dirichlet boundary condition

$$u(0) = 0 \tag{3.17}$$

Similarly, taking $c_L \longrightarrow \infty$ and setting $d_L = 0$ reduces (3.3) to

$$u(L) = 0. \tag{3.18}$$

Let

$$V = \{v(x) : \|v\|, \|v'\| < \infty, v(0) = v(L) = 0\}.$$

The boundary value problem consisting of equations (3.16) to (3.18) has variational formulation: Find $u \in V$ such that

$$\int_0^L pu'v' = \int_0^L fv, \forall v \in V \tag{3.19}$$

Note that the function u that solves (3.19) need not be as smooth as the function u that solves (3.1). Consequently, the former u is called a weak solution. Next, we prove existence and uniqueness of such a solution for the simple case introduced in this section. First recall the following theorem,

Theorem 3 (First vanishing theorem) *Let D be a bounded interval on $\mathbb{R}$ and let a function $f(x)$ be continuous on D. If $f(x) \geqslant 0$ and $\int_{\bar{D}} f(x)dx = 0$, then $f(x) = 0$ in $\bar{D} := D \cup \partial D$ where ∂D is the boundary of D.*

Theorem 4 *Solution u to (3.19) exists and is unique.*

Proof 6 *Assume that $u_1, u_2 \in V$ are solutions of (3.19) such that $u_1 \neq u_2$, then we have*

$$\int_0^L pu_1'v' = \int_0^L fv, \forall v \in V \tag{3.20}$$

and

$$\int_0^L pu_2'v' = \int_0^L fv, \forall v \in V \tag{3.21}$$

Taking the same $v \in V$ in (3.20) and (3.21), then subtracting equation (3.21) from equation (3.20) we get

$$\int_0^L p(u_1 - u_2)'v' = 0, \forall v \in V \tag{3.22}$$

If we let $v = u_1 - u_2 \in V$, then equation (3.22) becomes

$$\int_0^L p(u_1 - u_2)'^2 = 0 \tag{3.23}$$

Since $p(x) > 0$, it follows from the first vanishing theorem that

$$(u_1 - u_2)' = 0$$

in $[0, L]$ which upon integrating and substituting boundary conditions gives

$$u_1 = u_2$$

This contradicts the assumption that $u_1 \neq u_2$. Therefore, the weak solution to the variational formulation is unique. Since this solution is constructed using the solution of a linear system, existence follows from uniqueness.

3.5.2 A more general problem

In this section we prove the existence and uniqueness of a weak solution for the following boundary value problem

$$-(pu')' + cu = f, x \in I = [0, L], \tag{3.24}$$

$$u(0) = u(L) = 0 \tag{3.25}$$

where $c > 0$. Before we do that, we introduce some essential concepts.

Preliminaries

Definition 1 (Vector space) *A real vector space V is a set with operations $+ : V \times V \to V$ and $\cdot : \mathbb{R} \times V \to V$ such that, for all elements $u, v, w \in V$ also called vectors, and scalars $\lambda, \mu \in \mathbb{R}$, we have*

1. $u + v = v + u$
2. $(u + v) + w = u + (v + w)$
3. $\lambda(u + v) = \lambda u + \lambda v$
4. $(\lambda + \mu)u = \lambda u + \mu u$

Furthermore, there is a zero vector 0 such that $u + 0 = u$ and there exists a negative vector $-u$ such that $u + (-u) = 0$.

Definition 2 (Norm) *A norm on a vector space V is a mapping $\|\cdot\| : V \to \mathbb{R}$ such that the following properties are satisfied*

1. $\|u + v\| \leqslant \|u\| + \|v\|$ *(triangle inequality)*
2. $\|\lambda u\| = |\lambda|\|u\|$
3. $\|u\| \geqslant 0$, *with equality holding if and only if* $u = 0$

for all $u, v \in V$ and $\forall \lambda \in \mathbb{R}$.

Definition 3 (Normed vector space) *A normed vector space is a vector space equipped with a norm.*

Definition 4 (Linear form) *A linear form on a vector space V is a mapping $l(\cdot) : V \to \mathbb{R}$ that satisfies*

$$l(\alpha u + \beta v) = \alpha l(u) + \beta l(v), \ \forall u, v \in V \text{ and } \forall\, \alpha, \beta \in \mathbb{R} \tag{3.26}$$

Definition 5 *A linear form is said to be continuous on V, if there exist a constant C such that*

$$|l(v)| \leqslant C\|v\|, \ \forall v \in V \tag{3.27}$$

Definition 6 *Let V be a vector space, and let $u, v, w \in V$ and let $\alpha, \beta \in \mathbb{R}$, then*

i. A bilinear form is a mapping $a(\cdot\,,\cdot\,) : V \times V \to \mathbb{R}$ such that

(a) $a(\alpha u + \beta v, w) = \alpha a(u, w) + \beta a(v, w)$

(b) $a(u, \alpha v + \beta w) = \alpha a(u, v) + \beta a(u, w)$

ii. The bilinear form is symmetric if

$$a(u, v) = a(v, u) \tag{3.28}$$

iii. The bilinear form is continuous if there is a constant C such that

$$|a(u, v)| \leqslant C\|u\|\|v\| \tag{3.29}$$

Definition 7 (Inner product) *An inner product is a symmetric bilinear form $a(\cdot\,,\cdot\,)$ such that $a(u, u) \geqslant 0$, with equality holding if and only if $u = 0$, $\forall u \in V$.*

Definition 8 (Inner product space) *An inner product space is a vector space equipped with an inner product.*

In order to define a Hilbert space, we need to first understand what completeness means. Recall that a Cauchy sequence in a normed vector space V is a sequence $\{v_i\}_{i=0}^{\infty}$ of elements $v_i \in V, i = 1, 2...$, which for all $\epsilon > 0$ there is a positive integer n such that

$$\|v_i - v_j\| \leqslant \epsilon \; for \; i, j \geqslant n. \tag{3.30}$$

A sequence $\{v_i\}_{i=1}^{\infty}$ is convergent if there exist $v \in V$ such that for all $\epsilon > 0$ there is a positive integer n such that

$$\|v - v_i\| \leqslant \epsilon, \; for \; i \geqslant n \tag{3.31}$$

Definition 9 *A vector space is said to be complete if every Cauchy sequence in it is also convergent.*

Definition 10 (Hilbert space) *A complete inner product space is called a Hilbert space.*

Definition 11 *Let V be a Hilbert space and let $a(\cdot\,,\cdot\,) : V \times V \to \mathbb{R}$ be a bilinear form. We say that $a(\cdot\,,\cdot\,)$ is coercive if $\forall v \in V$, there exists a constant m such that*

$$m\|v\|_V^2 \leqslant a(v,v) \tag{3.32}$$

Theorem 5 (Cauchy-Schwarz Inequality) *Let V be a vector space equipped with an inner product $(\cdot\,,\cdot\,)$, then*

$$|(u,v)| \leqslant \|u\|\|v\|,\ \forall u,v \in V \tag{3.33}$$

Theorem 6 (Lax-Milgram Lemma) *Let V be a Hilbert space with inner product $(\cdot\,,\cdot\,)$, let $a(\cdot\,,\cdot\,)$ be a coercive and continuous bilinear form on V, and let $l(\cdot\,)$ be a continuous linear form on V. Then, there exist a unique solution $u \in V$ to the variational problem: Find $u \in V$ such that*

$$a(u,v) = l(v),\ \forall v \in V \tag{3.34}$$

3.5.3 Application of the Lax-Milgram Lemma

Poisson equation

Consider the boundary value problem consisting of equations (3.16) to (3.18). This problem has variational formulation: Find $u \in V$ such that

$$A(u,v) = l(v),\ \forall v \in V \tag{3.35}$$

where

$$A(u,v) := (pu',v') = \int_0^L pu'v',$$
$$l(v) := (f,v) = \int_0^L fv$$

and $V = H_0^1$ with

$$H_0^1 = \{w : [0,L] \to \mathbb{R} |\ \|w\|_V^2 := \|w'\|^2 + \|w\|^2 < \infty,\ w(0) = w(L) = 0\} \tag{3.36}$$

as a Hilbert space. We will make use of Lax-Milgram lemma to prove existence and uniquenes of solution for (3.35). We define the norm $\|\cdot\|$ by

$$\|w\| \equiv \|w\|_{L^2[0,L]} = \left(\int_0^L w^2\right)^{\frac{1}{2}} \equiv \sqrt{(w,w)}$$

Linearity of $l(\cdot)$ follows from

$$l(\alpha u_1 + \beta u_2) := (f, \alpha u_1 + \beta u_2) \quad = \alpha(f,u_1) + \beta(f,u_2) =: \alpha l(u_1) + \beta l(u_2)$$

The bilinearity of $A(\cdot\,,\cdot\,)$ follows from

$$A(\alpha u_1 + \beta u_2, v) := (p(\alpha u_1 + \beta u_2)', v') \quad = \alpha(pu_1', v') + \beta(pu_2', v') =: \alpha A(u_1, v) + \beta A(u_2, v)$$

and a similar argument follows for the second slot. Continuity of $l(\cdot\,)$ follows from using the *Cauchy-Schwarz (C-S) inequality* as follows

$$\begin{aligned}
|l(v)| := |(f, v)| &\leqslant \|f\|\|v\| \\
&\leqslant \|f\|(\|v\| + \|v'\|) && (\|\cdot\| \geqslant 0) \\
&\leqslant \sqrt{2}\|f\|(\|v\|^2 + \|v'\|^2)^{\frac{1}{2}} && \left(\sum_{i=1}^{n} a_i b_i \leqslant \left(\sum_{i=1}^{n} a_i^2\right)^{\frac{1}{2}} \left(\sum_{i=1}^{n} b_i^2\right)^{\frac{1}{2}}\right) \\
&= C\|v\|_V
\end{aligned}$$

The continuity of $A(\cdot\,,\cdot\,)$ follows from the discrete C-S inequality as shown below

$$\begin{aligned}
|A(u, v)| := |(pu', v')| \\
&\leqslant \|pu'\|\|v'\| \\
&\leqslant \|p\|_\infty \|u'\|\|v'\| && (\|p\|_\infty := \max_{0\leqslant x\leqslant L} |p(x)|) \\
&\leqslant \|p\|_\infty (\|u'\|\|v'\| + \|u\|\|v\|) && (0 \leqslant \|\cdot\|) \\
&\leqslant \|p\|_\infty (\|u'\|^2 + \|u\|^2)^{\frac{1}{2}} (\|v'\|^2 + \|v\|^2)^{\frac{1}{2}} && \text{(discrete C-S inequality)} \\
&= C\|u\|_V\|v\|_V,
\end{aligned}$$

where $C = \|p\|_\infty$. Lastly, we show that $A(\cdot\,,\cdot\,)$ is coercive as follows

$$\begin{aligned}
A(u, u) &= (pu', u') \\
&\geqslant \bar{p}\|u'\|^2 && (p(x) > 0 \text{ and } \bar{p} := \min_{0\leqslant x\leqslant L} p(x)) \\
&= C\|u'\|^2
\end{aligned}$$

where $C = \bar{p}$. By the Lax-Milgram lemma, it follows that the variational formulation (3.35) has a unique solution.

Reaction-diffusion equation

As a second example of applying Lax-Milgram Lemma, we consider the boundary value problem consisting of equations (3.24) and (3.25), where $c > 0$. This problem has the variational formulation: Find $u \in V$ such that

$$A(u, v) = l(v), \ \forall \ v \in V \tag{3.37}$$

where

$$A(u, v) := (pu', v') + (cu, v) = \int_0^L pu'v' + \int_0^L cuv,$$

$$l(v) := (f, v) = \int_0^L fv$$

and $V = H_0^1$ is the Hilbert space shown in equation (3.36).

The linearity of $l(.)$ follows from the fact that integration is linear.
The continuity of $l(.)$ was already proved for the Poisson equation.
The bilinearity of $A(\cdot\,,\cdot\,)$ follows from linearity of the derivatives and linearity of the integral.
The continuity of $A(\cdot\,,\cdot\,)$ follows from

$$
\begin{aligned}
|A(u,v)| &\leq |(pu',v') + (cu,v)| && \text{(Triangle inequality)}\\
&\leqslant \|pu'\|\|v'\| + \|cu\|\|v\| && \text{(Cauchy-Schwarz inequality)}\\
&\leqslant \|p\|_\infty\|u'\|\|v'\| + \|c\|_\infty\|u\|\|v\| && (\|c\|_\infty := \min_{0\leqslant x\leqslant L} c(x))\\
&\leqslant C(\|u'\|\|v'\| + \|u\|\|v\|) && (C = \max\{\|p\|_\infty, \|c\|_\infty\})\\
&\leqslant C(\|u'\|^2 + \|u\|^2)^{\frac{1}{2}}(\|v'\|^2 + \|v\|^2)^{\frac{1}{2}} && \text{(discrete Cauchy-Schwarz inequality)}\\
&= C\|u\|_V\|v\|_V
\end{aligned}
$$

Lastly, the coercivity of $A(\cdot\,,\cdot\,)$ follows from

$$
\begin{aligned}
A(u,u) &:= (pu',u') + (cu,u)\\
&\geqslant \bar{p}\|u'\|^2 + \bar{c}\|u\|^2 && ((c>0, \bar{c} := \min_{0\leqslant x\leqslant L} c(x)) \text{ and } \bar{p} := \min_{0\leqslant x\leqslant L} p(x))\\
&\geqslant C(\|u'\|^2 + \|u\|^2) && \text{where } (C = \min\{\bar{p},\bar{c}\})\\
&= C\|u\|_V^2 && (3.38)
\end{aligned}
$$

By the Lax-Milgram lemma, variational formulation (3.37) has a unique solution.

3.6 Computer implementation for original problem

Let $m_i = \frac{x_i + x_{i-1}}{2}$ be the mid-point of each subinterval $I_i = [x_{i-1}, x_i]$.

Recall the following quadrature rules:

1. Mid-point rule
$$\int_{I_i} f(x)dx \approx f(m_i)h_i \tag{3.39}$$
2. Trapezoidal rule
$$\int_{I_i} f(x)dx \approx \frac{f(x_{i-1}) + f(x_i)}{2}h_i \tag{3.40}$$

3. Simpson's rule

$$\int_{I_i} f(x)dx \approx \frac{f(x_{i-1}) + 4f(m_i) + f(x_i)}{6} h_i \tag{3.41}$$

where $h_i = x_i - x_{i-1}$.

3.6.1 Assembly of stiffness matrix

Since

$$A_{rs} = \int_0^L p\varphi'_r\varphi'_s = \sum_{i=1}^N \int_{x_{i-1}}^{x_i} p\varphi'_r\varphi'_s = \sum_{i=1}^N A^i_{rs} \tag{3.42}$$

then

$$A = \sum_{i=1}^N A^i \tag{3.43}$$

From the definition of φ_i we have that

$$\varphi'_i(x) = \begin{cases} \frac{1}{h_i} & \text{on } I_i \\ \frac{1}{h_{i+1}} & \text{on } I_{i+1} \\ 0, & \text{elsewhere} \end{cases}$$

which implies that, all the entries of A are zero except $A_{i-1,i-1}$, $A_{i-1,i}$, $A_{i,i-1}$ and $A_{i,i}$ and

$$A^i_{rs} = \begin{cases} \int_{x_{i-1}}^{x_i} p_r\varphi'_r\varphi'_s \neq 0, & r = i-1 \text{ or } i \text{ and } s = i-1 \text{ or } i \\ 0, & \text{otherwise} \end{cases}$$

Making use of the Mid-point rule gives:

$$A^i_{i-1,i-1} = \int_{x_{i-1}}^{x_i} p_i\varphi'^2_{i-1} \approx p_i\left(\frac{-1}{h_i}\right)^2 h_i = \frac{p_i}{h_i}$$

where $p_i = p(m_i)$.

Also,

$$A^i_{i-1,i} = \int_{x_{i-1}}^{x_i} p_i\varphi'_{i-1}\varphi'_i \approx p_i\left(\frac{-1}{h_i}\right)\left(\frac{1}{h_i}\right) h_i = -\frac{p_i}{h_i} \tag{3.44}$$

By symmetry of the inner product, we have that

$$A^i_{i,i-1} = -\frac{p_i}{h_i}$$

Therefore, all non-zero terms of A^i are contained in the *local element stiffness matrix*

$$\begin{pmatrix} A^i_{i-1,i-1} & A^i_{i-1,i} \\ A^i_{i,i-1} & A^i_{i,i} \end{pmatrix} = \frac{p_i}{h_i}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

which is part of the *global element stiffness matrix*:

$$A^i = \frac{p_i}{h_i} \begin{array}{c} \\ \left(\begin{array}{cccc} & i-1 & i & \\ & & & \\ & 1 & -1 & \\ & -1 & 1 & \\ & & & \end{array}\right) \end{array} \begin{array}{c} \\ \\ i-1 \\ i \\ \\ \end{array}$$

where the only non-zero terms are in rows $i-1$ and i, and in columns $i-1$ and i. If we pad the local element stiffness matrix with zeros to form the global element stiffness matrix A^i then we construct A using equation (3.43). A more practical way is the assembly procedure. In this case we add the 2×2 local element stiffness matrices to stiffness matrix A which is originally filled with zeros to get

$$A = \begin{pmatrix} \frac{p_1}{h_1} & -\frac{p_1}{h_1} & & & \\ -\frac{p_1}{h_1} & \frac{p_1}{h_1}+\frac{p_2}{h_2} & -\frac{p_2}{h_2} & & \\ & -\frac{p_2}{h_2} & \frac{p_2}{h_2}+\frac{p_3}{h_3} & -\frac{p_3}{h_3} & \\ & & \ddots & \ddots & \ddots \\ & & -\frac{p_{n-1}}{h_{n-1}} & \frac{p_{n-1}}{h_{n-1}}+\frac{p_n}{h_n} & -\frac{p_n}{h_n} \\ & & & -\frac{p_n}{h_n} & \frac{p_n}{h_n} \end{pmatrix}$$

in the appropriate rows and columns. Algorithm 2 highlights all the necessary steps for assembling A.

Algorithm 2: Assemble stiffness matrix

```
1 Initialize $A \in \mathbb{R}^{(n+1)\times(n+1)}$ as a zero matrix.
2 for $i = 1, 2, \cdots, n$ do
3     Compute local element stiffness matrix
          $$A^{I_i} = \frac{p_i}{h_i}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$
      where $p_i = p((x_{i-1} + x_i)/2)$ and $h_i = x_i - x_{i-1}$.
4     Add $A^{I_i}_{11}$ to $M_{ii}$.
5     Add $A^{I_i}_{12}$ to $M_{i,i+1}$.
6     Add $A^{I_i}_{21}$ to $M_{i+1,i}$.
7     Add $A^{I_i}_{22}$ to $M_{i+1,i+1}$.
8 end
```

This algorithm translates to the MATLAB code shown in Listing 1.

Listing 3.1: MATLAB code for assembling stiffness matrix

```
function A=StiffnessAssembler1D(x,p)
    n=length(x)-1; % number of elements
    A=zeros(n+1,n+1); % initialise stiffness matrix
    for i=1:n
```

```
        h=x(i+1)-x(i); % element length
        pmid=p((x(i+1)+x(i))/2);
        A(i,i)=A(i,i)+pmid/h;
        A(i,i+1)=A(i,i+1)-pmid/h;
        A(i+1,i)=A(i+1,i)-pmid/h;
        A(i+1,i+1)=A(i+1,i+1)+pmid/h;
    end
```

Input arguments for this routine are vector x containing nodes $x_0, x_1, \ldots, x_n$, and function p (assumed to be a separate routine) liable for passing function $p(x)$ to the function StiffnessAssembler1D. Output from this routine is the stiffness matrix A.

3.6.2 Assembly of mass matrix

Since

$$M_{rs} = \int_0^L c\varphi_r\varphi_s = \sum_{i=1}^N \int_{x_{i-1}}^{x_i} c\varphi_r\varphi_s = \sum_{i=1}^N M_{rs}^i \tag{3.45}$$

then

$$M = \sum_{i=1}^N M^i \tag{3.46}$$

where M is the global element mass matrix. On $[x_{i-1}, x_i]$, only φ_{i-1},$\varphi_i \neq 0$. Consequently,

$$M_{rs}^i = \begin{cases} \int_{x_{i-1}}^{x_i} c\varphi_r\varphi_s \neq 0, & r = i-1 \text{ or } i \text{ and } s = i-1 \text{ or } i \\ 0, & \text{otherwise} \end{cases}$$

Now, we compute the non-zero terms of M_i.

$$\begin{aligned} M_{i-1,i-1}^i &= \int_{x_{i-1}}^{x_i} c\varphi_{i-1}^2 \\ &\approx c_i \left(\frac{(\varphi_{i-1}(x_{i-1}))^2 + 4(\varphi_{i-1}(m_i))^2 + (\varphi_{i-1}(x_i))^2}{6} h_i \right) \\ &= c_i \left(\frac{1 + 4(\frac{1}{2})^2 + 0}{6} h_i \right) = \frac{c_i h_i}{3} \end{aligned} \tag{3.47}$$

where we have used Simpson's rule to estimate the integral. Similarly,

$$M_{i,i}^i = \frac{c_i h_i}{3}$$

Also

$$M_{i-1,i}^i = \int_{x_{i-1}}^{x_i} c\varphi_{i-1}\varphi_i \approx \frac{c_i(1(0) + 4(\frac{1}{2})^2 + 0(1))}{6} h_i = \frac{c_i h_i}{6} = M_{i,i-1}^i \tag{3.48}$$

by the symmetry of $(\cdot\,,\cdot)$. Therefore, all non-zero terms of M^i are contained in the *local element mass matrix*

$$\begin{pmatrix} M_{i-1,i-1}^i & M_{i-1,i}^i \\ M_{i,i-1}^i & M_{i,i}^i \end{pmatrix} = \frac{c_i h_i}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

The following algorithm summarizes the assembly procedure for constructing M.

Algorithm 3: Assemble mass matrix

```
1 Initialize $A \in \mathbb{R}^{(n+1)\times(n+1)}$ as a zero matrix.
2 for $i = 1, 2, \cdots, n$ do
3     Compute local element stiffness matrix
          $$M^{I_i} = \frac{c_i h_i}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$
      where $h_i = x_i - x_{i-1}$.
4     Add $M^{I_i}_{11}$ to $M_{ii}$.
5     Add $M^{I_i}_{12}$ to $M_{i,i+1}$.
6     Add $M^{I_i}_{21}$ to $M_{i+1,i}$.
7     Add $M^{I_i}_{22}$ to $M_{i+1,i+1}$.
8 end
```

Algorithm 3 translates to MATLAB code in Listing 3.2.

Listing 3.2: `MATLAB` code to assemble mass matrix

```matlab
function M = MassAssembler1D(x,c)
    n = length(x) - 1; % number of elements
    M = zeros(n+1);  % initialise mass matrix
    for i = 1 : n
        h = x(i+1)-x(i); % element length
        cmid=c((x(i+1)+x(i))/2); % c at the midpoint
        M(i,i) = M(i,i) + cmid*h/3;
        M(i,i+1) = M(i,i+1) + cmid*h/6;
        M(i+1,i) = M(i+1,i) + cmid*h/6;
        M(i+1,i+1) = M(i+1,i+1) + cmid*h/3;
    end
```

The function MassAssembler1D takes in two parameters: vector x containing the nodes $x_0, x_1, \ldots, x_n$ and a user defined function for computing $c(x)$. Output from this routine is the global mass matrix M.

3.6.3 Assembly of load vector

By linearity of the integral we have that

$$b = \sum_{i=1}^{N} b^i \tag{3.49}$$

where

$$b_r^i = \begin{cases} \int_{x_{i-1}}^{x_i} f\varphi_r \neq 0, & r = i-1, i \\ 0, & \text{otherwise} \end{cases}$$

since the *hat functions* have small support. We compute non-zero terms of b below.

$$b_{i-1}^i = \int_{x_{i-1}}^{x_i} f\varphi_{i-1} \approx \frac{f(x_{i-1})\varphi_{i-1}(x_{i-1}) + f(x_i)\varphi_{i-1}(x_i)}{2} h_i = \frac{f(x_{i-1})h_i}{2} \tag{3.50}$$

where we have used the Trapezoidal rule for estimating the integral. Similarly

$$b_i^i = \int_{x_{i-1}}^{x_i} f\varphi_i \approx \frac{f(x_{i-1})\varphi_i(x_{i-1}) + f(x_i)\varphi_i(x_i)}{2} h_i = \frac{f(x_i)h_i}{2} \tag{3.51}$$

Therefore, all non-zero terms of b^i are contained in local element load vector

$$\begin{pmatrix} b_{i-1}^i \\ b_i^i \end{pmatrix} = \frac{h_i}{2} \begin{pmatrix} f(x_{i-1} \\ f(x_i) \end{pmatrix}$$

The assembly procedure for load vector is shown in Algorithm 4, which translates to the MATLAB code in Listing 3.3.

Algorithm 4: Assembling load vector

```
1 Initialise $b \in \mathbb{R}^{(n+1)\times 1}$ as a zero matrix.
2 for $i = 1, 2, \cdots, n$ do
3     Compute local element load vector
          $$b^{I_i} = \frac{h_i}{2} \begin{pmatrix} f(x_{i-1}) \\ f(x_i) \end{pmatrix}$$
       where $h_i = x_i - x_{i-1}$.
4     Add $b_1^{I_i}$ to $b_{i-1}$.
5     Add $b_2^{I_i}$ to $b_i$.
6 end
```

Listing 3.3: `MATLAB` code to assemble load vector

```matlab
function b = LoadAssembler1D(x,f)
    n = length(x) - 1; % number of elements
    b = zeros(n+1,1); % initialise load vector
    for i = 1:n
        h = x(i+1) - x(i); % element length
        b(i) = b(i) + h*f(x(i))/2;
        b(i+1) = b(i+1) + h*f(x(i+1))/2;
    end
```

The routine LoadAssembler1D takes in two input arguments, a vector x holding the nodal coordinates and function f which is also a routine on its own. Output from LoadAssembler1D is the load vector b.

3.6.4 Assembly of boundary matrices

Since

$$\varphi_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

then

$$R_{ij} = c_L \varphi_j(L)\varphi_i(L) + c_0 \varphi_j(0)\varphi_i(0) = c_L \delta_{jL}\delta_{in} + c_0 \delta_{j0}\delta_{i0}$$

and $R_{ij} \neq 0$ only when either $i = j = 0$ or $i = j = n$. Consequently, $R_{00} = c_0$ and $R_{nn} = c_L$. Therefore,

$$R = \begin{pmatrix} c_0 & & \\ & \ddots & \\ & & c_L \end{pmatrix}$$

The routine StiffnessAssembler1D for assembling the *stiffness matrix* is modified to include the *boundary matrix R* as shown in Listing 3.4.

Listing 3.4: MATLAB code for assembling stiffness matrix and including boundary terms

```
function A=StiffnessAssembler1D(x,p,c)
    n=length(x)-1; % number of elements
    A=zeros(n+1,n+1); % initialise stiffness matrix
    for i=1:n
        h=x(i+1)-x(i); % element length
        pmid=p((x(i+1)+x(i))/2);
        A(i,i)=A(i,i)+pmid/h;
        A(i,i+1)=A(i,i+1)-pmid/h;
        A(i+1,i)=A(i+1,i)-pmid/h;
        A(i+1,i+1)=A(i+1,i+1)+pmid/h;
    end
    A(1,1) = A(1,1) + c(1); % c_0
    A(n+1,n+1) = A(n+1,n+1) + c(2); % c_L
```

Output from this routine is the matrix $A + R$.
We compute matrix r in a similar manner to that for R. Since

$$r_i = (c_L d_L + e_L)\varphi_i(L) + (c_0 d_0 + e_0)\varphi_i(0) = (c_L d_L + e_L)\delta_{in} + (c_0 d_0 + e_0)\delta_{i0},$$

then $r_i \neq 0$ only when either $i = 0$ or $i = n$. Therefore, $r_0 = c_0 d_0 + e_0$ and $r_n = c_L d_L + e_L$, and

$$r = \begin{pmatrix} c_0 d_0 + e_0 \\ \vdots \\ c_L d_L + e_L \end{pmatrix}$$

Similarly, the routine LoadAssembler1D is modified to include the matrix r as shown in Listing 3.5.

Listing 3.5: `MATLAB` code to assemble load vector

```
function b = LoadAssembler1D(x,f,c,d,e)
    n = length(x) - 1; % number of elements
    b = zeros(n+1,1); % initialise load vector
    for i = 1:n
        h = x(i+1) - x(i); % element length
        b(i) = b(i) + h*f(x(i))/2;
        b(i+1) = b(i+1) + h*f(x(i+1))/2;
    end
    b(1) = b(1) + c(1)*d(1) + e(1);  % c_0 d_0 + e_0
    b(n+1) = b(n+1) +  c(2)*d(2) + e(2); % c_L d_L + e_L
```

This routine takes an additional input argument vector a for including the boundary terms and produces the matrix $b + r$.

3.6.5 Putting it all together

Example 1

Consider the problem of finding the approximate solution to the boundary value problem

$$-u'' + u = x, \quad x \in [0, 1] \tag{3.52}$$

$$u(0) = 0, u(1) = 0 \tag{3.53}$$

Solution

In order to obtain equation (3.52) from the governing equation (3.1), we set $p(x) = c(x) = 1$, and $f(x) = x$. Boundary conditions (3.53) result from (3.2) and (3.3) if we set $c_L = c_0 = 10^6$, $d_L = d_0 = 0$, and $e_L = e_0 = 1$.

Listing 3.6: `MATLAB` main solver routine

```
function ReactionDiffusionSolver1D() % no input, no output
   h=0.01; % element length
   x=0:h:1; % mesh
   c=[10^6 10^6]; % c_0, c_L
   K=StiffnessAssembler1D(x,@pfun,c);
   M=MassAssembler1D(x,@qfun);
   d=[0 0]; % d_0, d_L
   e=[1 1]; % e_0, e_L
   b=LoadAssembler1D(x,@ffun,c,d,e);
   u=(K+M)\b; % solve linear system
   plot(x,u); xlabel('x'); ylabel('u')
function y=pfun(x)
   y=1; % p(x)
function y=qfun(x)
   y=1; % q(x)
function y=ffun(x)
```

```
            y=x; % f(x)
```

The MATLAB routine in Listing 3.6 together with routines in Listings 3.2, 3.4, and 3.5 generate the finite element approximation u_h shown in Figure 3.1 that makes use of a mesh with 100 elements. The solution u_h satisfies the Dirichlet boundary condition at $x = 0$ and $x = 1$ as shown in Figure 3.1.

Differential equation (3.52) subject to boundary condition (3.53) has exact solution

$$u(x) = x - \frac{\sinh x}{\sinh 1}. \tag{3.54}$$

Figure 3.1 shows good agreement between the exact and approximate solutions

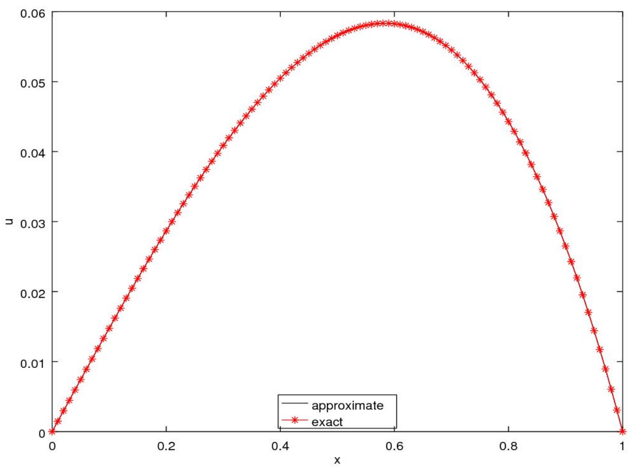


Figure 3.1: Comparison of the approximate solution with the exact solution for problem (3.52) and (3.53).

Let

$$\|e\|_\infty := \max_{1\le i\le n} |e(x_i)|,$$

and let

$$\|e\|_{L^2(I)} := \left(\int_I |e(x)|^2 dx\right)^{\frac{1}{2}}$$

where $e(x) = u(x) - u_h(x)$.

Recall from the theory that

$$\|u - u_h\|_{L^2(I)} = \|e\|_{L^2(I)} \le Ch^2$$

as $h \to 0$ and for some constant $C > 0$. Hence, the finite element solution u_h is second order accurate and we write

$$\|e\|_{L^2(I)} = O(h^2).$$

Table 3.1 shows that as the number of elements n increases, i.e as we refine the mesh, the size of each element $h \to 0$. It also suggests that

$$\|e\|_\infty / h^2 \leq 4.422 \times 10^{-3} = C$$

as $h \to 0$

n	$h = 1/n$	$\|e\|_\infty$	$\|e\|_\infty / h^2$
4	0.250000	2.698×10^{-4}	4.303×10^{-3}
8	0.125000	6.885×10^{-5}	4.406×10^{-3}
16	0.062500	1.722×10^{-5}	4.409×10^{-3}
32	0.031250	4.319×10^{-6}	4.422×10^{-3}
64	0.015625	1.080×10^{-6}	4.422×10^{-3}
128	0.007812	2.699×10^{-7}	4.422×10^{-3}

Table 3.1: Maximum errors of the finite element solution for problem (3.52) and (3.53).

The integral $\|e\|_{L^2(I)}$ in Table 3.2 is evaluated numerically using Trapezoidal rule since it is generally difficult to evaluate analytically. Table 3.2 shows that

$$\|e\|_{L^2(I)} / h^2 \leq 3.170 \times 10^{-3} = C$$

as $h \to 0$, which is in agreement with the theory in Chapter 2.

n	$h = 1/n$	$\|e\|_{L^2(I)}$	$\|e\|_{L^2(I)} / h^2$
4	0.250000	1.990×10^{-4}	3.185×10^{-3}
8	0.125000	4.962×10^{-5}	3.175×10^{-3}
16	0.062500	1.239×10^{-5}	3.171×10^{-3}
32	0.031250	3.096×10^{-6}	3.170×10^{-3}
64	0.015625	7.739×10^{-7}	3.170×10^{-3}
128	0.007812	1.935×10^{-7}	3.170×10^{-3}

Table 3.2: L^2 norm errors of the finite element solution for problem (3.52) and (3.53).

3.7 Adaptive finite element method

Model problem

Consider the simple model problem (3.16) - (3.18) where $p \equiv 1$, then we have the following error estimate.

Theorem 7 (Error estimate) *The finite element solution $u_h \in V_h$ such that*

$$\int_I u_h \varphi_i = \int_I f\varphi_i, \quad i = 1, 2, \ldots, n-1, \tag{3.55}$$

satisfies the estimate

$$\|(u-u_h)'\|^2_{L^2(I)} \le C\sum_{i=1}^{n} \eta_i^2(u_h) \tag{3.56}$$

where

$$\eta_i(u_h) \quad = \quad h_i\|f+u_h''\|_{L^2(I_i)} = h_i\|f\|_{L^2(I_i)} \qquad (\textit{since } u_h''|_{I_i} = 0)$$

and $I = \cup_{i=1}^{n} I_i$ is disjoint.

Proof 7 *Let $e = u - u_h$, then*

$$\begin{aligned}
\|e'\|^2_{L^2(I)} &= \int_I e'^2 dx \\
&= \int_I e'(e-\pi e)'dx && (\textit{Galerkin orthogonality}) \\
&= \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} e'(e-\pi e)'dx \\
&= \sum_{i=1}^{n} e'(e-\pi e)|_{x_{i-1}}^{x_i} - \int_{x_{i-1}}^{x_i} e''(e-\pi e)dx && (\textit{Integrating by parts}) \\
&= \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} -e''(e-\pi e)dx && (e = \pi e \textit{ at the nodes}) \\
&= \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} (-u''+u_h'')(e-\pi e)dx && (e = u-u_h) \\
&= \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} (f+u_h'')(e-\pi e)dx && (-u'' = f) \\
&\le \sum_{i=1}^{n} \|(f+u_h'')\|_{L^2(I_i)}\|(e-\pi e)\|_{L^2(I_i)} && (\textit{Triangle inequality}) \\
&\le C\sum_{i=1}^{n} h_i\|(f+u_h'')\|_{L^2(I_i)}\|e'\|_{L^2(I_i)} && (\|e\|_{L^2(I)} \le h\|e'\|_{L^2(I)} \\
&\le C\left(\sum_{i=1}^{n} h_i^2\|(f+u_h'')\|^2_{L^2(I_i)}\right)^{\frac{1}{2}}\left(\sum_{i=1}^{n}\|e'\|^2_{L^2(I_i)}\right)^{\frac{1}{2}} && \left(\sum_{i=0}^{n} a_i b_i \leqslant \left(\sum_{i=0}^{n} a_i^2\right)^{\frac{1}{2}}\left(\sum_{i=0}^{n} b_i^2\right)^{\frac{1}{2}}\right)
\end{aligned}$$

Now, replacing $\left(\sum_{i=1}^{n}\|e'\|^2_{L^2(I_i)}\right)^{\frac{1}{2}}$ by $\|e'\|_{L^2(I)}$ and dividing both sides by $\|e'\|_{L^2(I)}$ we get

$$\|e'\|_{L^2(I)} \le C\left(\sum_{i=1}^{n} h_i^2\|(f+u_h'')\|^2_{L^2(I_i)}\right)^{\frac{1}{2}}. \tag{3.57}$$

Adaptive mesh refinement

Recall from Chapter 2 that

$$\|e\|_{L^2(I)} \leq C\|e'\|_{L^2(I)}$$

for some constant C. It follows from inequality (3.56) that the size of the error depends on the element residual $\eta_i(u_h)$ on I_i, where $i = 1, 2, \cdots, n$. Therefore, it is essential to reduce the size of elements with largest residual to obtain more accurate solutions. This process of refining elements together with the finite element methods result in the adaptive finite element method and Algorithm 5 is the summary of this process.

Algorithm 5: Adaptive finite element method

```
1 Given a coarse mesh with $n$ elements.
2 **while** $n$ *is not large enough* **do**
3     Compute finite element solution $u_h$.
4     Evaluate element residuals
              $\eta_i(u_h) = h_i\|f + u_h''\|_{L^2(I_i)}; i = 1, 2, \ldots, n$
      where $h_i = x_{i+1} - x_i$ is the length of each element.
5     Select and refine the most error prone elements i.e with the largest
      residuals.
6 **end**
```

Algorithm 5 has four basic steps:

1. Computation of $\eta_i(u_h); i = 1, 2, \ldots, n$. η_i was evaluated using the Trapezoidal rule. Listing 3.7 shows how this step was implemented.

2. Selection of elements for refinement happens if

$$\eta_i > \alpha \max_{i=1,2,\ldots,n} \eta_i$$

   where $0 \leq \alpha \leq 1$ is a parameter to be chosen. Note that, if $\alpha = 0$ we get uniform refinement whereas $\alpha = 1$ gives no refinement.

3. Suppose element I_i was chosen for refinement. The refinement procedure consists of replacing I_i with

$$[x_i, (x_i + x_{i+1})/2] \cup [(x_i + x_{i+1})/2, x_{i+1}].$$

4. Stopping criteria for the algorithm take the form of a maximum bound on the number of elements, the memory usage, the total size of the residual, and so on.

Listing 3.7: `MATLAB` routine for computing element residuals

```
    function eta = residual(x,f)
        eta = zeros(1,n);
        for i = 1:n % loop over elements
            h = x(i+1) - x(i); % element length
            a = f(x(i)); % temporary variables
            b = f(x(i+1));
            %use Trapezoidal rule to evaluate the integral of f^2.
            t = (a^2+b^2)*h/2;
            eta(i) = h*sqrt(t); % element residual
         end
     end
```

The function residual in Listing 3.7 requires 2 inputs arguments x and f, and it produces one output argument η_i.

Listing 3.8: `MATLAB` routine highlighting the refinement procudure

```
  function x = refine(eta,x)
      alpha = 0.9; % allocate element residuals
      for i = 1:n % loop over elements
          if eta(i) > alpha*max(eta)
              x = [x  (x(i+1) + x(i))/2];
          end
      end
      x = sort(x);
  end
```

Basic steps 2 and 3 of Algorithm 5 translate to the MATLAB code in Listing 3.8. In basic step 4, we chose to use the maximum number N of nodes as a stopping criterion.

Example 2

Consider the problem of finding an approximate solution of the following boundary value problem

$$-u'' = f(x), \quad x \in [0,1] \tag{3.58}$$

$$u(0) = u(1) = 0 \tag{3.59}$$

where

$$f(x) = \begin{cases} 2x, \; x \in \left[0, \frac{1}{2}\right] \\ 2 - 2x, \; x \in \left[\frac{1}{2}, 1\right] \end{cases}$$

First we shall do this using the classical FEM, then we solve the problem using the adaptive FEM. We conclude by comparing results of the two methods.

Solution

Setting $p(x) = 1$ and $c(x) = 0$ in the governing equation (3.1) gives equation (3.58). Now, in order to obtain the boundary conditions (3.59) making use of the boundary conditions (3.2) and (3.3), we set $d_0 = d_L = 0, e_0 = e_L = 1, c_L = c_0 = 10^6$.

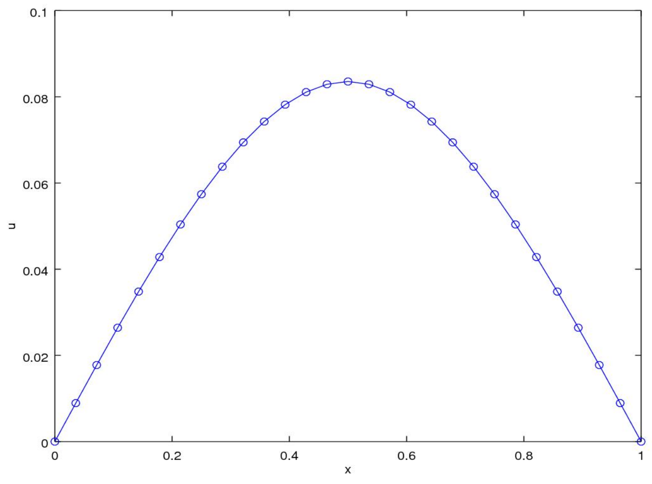


Figure 3.2: Approximate solution of the problem consisting of equations (3.58) and (3.59) using the classical FEM.

Listing 3.9: `MATLAB` code for the classical FEM solver.

```
function ReactionDiffusionSolver1D() % no input, no output
    x = linspace(0,1,29) % mesh with 22 elements
    c=[10^6 10^6]; % c 0, c L
    K=StiffnessAssembler1D(x,@pfun,c);
    M=MassAssembler1D(x,@qfun);
    d=[0 0]; % d 0, d L
    e=[1 1]; % e 0, e L
    b=LoadAssembler1D(x,@ffun,c,d,e);
    u=(K+M)\b; % solve linear system
    plot(x,u); xlabel('x'); ylabel('u')
end
function y=pfun(x)
    y=1; % p(x)
end
function y=qfun(x)
    y=0; % q(x)
end
function y=ffun(x)
    for i = 1:length(x)
       y(i)=2*x(i);
       if x(i) >= 0.5
          y(i) = 2 - 2*x(i);
       end
    end
end
```

The numerical solution u_h shown in Figure 3.2 was generated by the MATLAB routine in Listing 3.9 using a mesh with 28 elements or 29 equally spaced nodes.

The classical FEM was used in this case, this routine makes use of the routines in Listings 3.2, 3.4, and 3.5 to generate the mass matrix, the stiffness matrix and the load vector respectively. It is clear from Figure 3.2 that u_h satisfies the Dirichlet boundary condition at $x = 0$ and at $x = 1$. Figure 3.3 shows the adaptive finite element solution to the boundary value problem 3.58 .

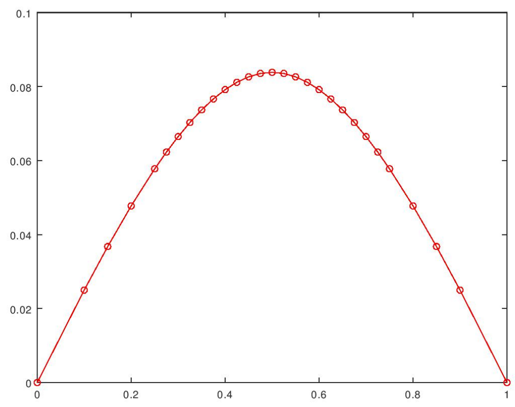


Figure 3.3: Adaptive finite element solution of problem (3.58) and (3.59).

The adaptive FEM turns to focus more on the peak. Figure 3.3 shows that the nodes on the peak are very close to each other with irregular spacing to make the geometry finer. The reason for doing this is to obtain high accurate results. In Listing 3.10 is the MATLAB routine used to acquire the results shown in Figure 3.3.

Listing 3.10: `MATLAB` code fot the adaptive FEM solver.

```
function n=AdaptiveReactionDiffusionSolver1D()
    x = 0:0.2:1; % start with a course mesh
    n = length(x); % number of nodes
    N = 23; % maximum number of nodes
    while 1
        u = ReactionDiffusionSolver1D(x);
        if n >= N
          break
        end
        eta = residual(x,@ffun);
        x = refine(eta,x);
        n = length(x);
   endwhile
   u=u';
   x;
   nodes=length(x)
   plot(x,u',x,u','-ro')
end
function u=ReactionDiffusionSolver1D(x) % no input, no output
    c=[10^6 10^6]; % c 0, c L
    K=StiffnessAssembler1D(x,@pfun,c);
    M=MassAssembler1D(x,@qfun);
    d=[0 0]; % d 0, d L
    e=[0 0]; % e 0, e L
    b=LoadAssembler1D(x,@ffun,c,d,e);
    u=(K+M)\b; % solve linear system
end
function y=pfun(x)
    y=1; % p(x)
end
function y=qfun(x)
    y=0; % q(x)
end
function y = ffun(x)
    for i=1:length(x)
      y(i)=2*x(i);
      if x(i) >= 0.5
        y(i)=2-2*x(i);
      end
    end
end
```

Shown in Listing 3.10 is the routine used to produce the results shown in Figure 3.3. According to Algorithm 5 we start with a coarse mesh consisting of 6 equally spaced nodes on $[0, 1]$ then, elements with the largest residuals are selected for refinement. This routine terminates once the number of nodes in the refined mesh on $[0, 1]$ reaches 23 otherwise only refine once beyond 23 nodes. For this experiment, the routine terminated once the number of nodes reached 29 and this is highly possible since more than 2 elements could be refined simultaneously.

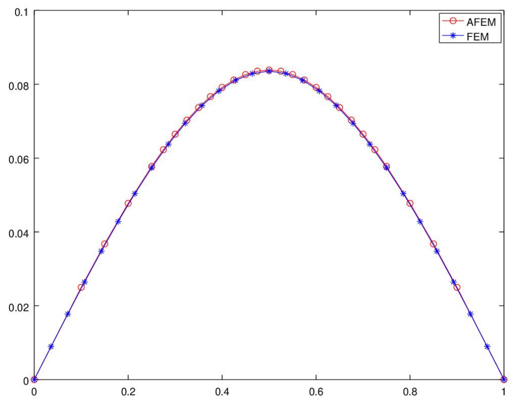


Figure 3.4: Comparison of the finite element solution with the Adaptive finite element solution of problem (3.58) and (3.59).

Figure 3.4 suggests that the finite element solution u_h and the adaptive finite element solution u_h^a are the same, but

$$\|u_h\|_\infty = 0.083844$$

and

$$\|u_h^a\|_\infty = 0.0835472$$

indicating that solutions are slightly different.

3.8 Summary

In this chapter we used the FEM to seek a numerical solution for the RDE in $\mathbb{R}$. We proved the existence and the uniqueness of the solution for a simple problem and a more general problem. We made used of Octave to implement the FEM. We also derived error estimates and used them to formulate the adaptive FEM. Lastly, we used two illustrative examples to investigate the computational efficiency of the method.

Chapter 4

Reaction-diffusion equation in the real plane

4.1 Introduction

In this chapter we develop a finite element method for numerically solving the reaction-diffusion equation in the real plane. We first convert the gorverning equation to its equivalent variational form and then we look for an approximate solution of the variational problem in the space of continuous piecewise linear functions. We also provide proofs for some error estimates. For application purposes MATLAB's pdetool [13] and Python's FEniCS [8] will be employed to find an approximate solution for a simple problem.

4.2 Model problem

Consider the problem of solving the reaction-diffusion equation

$$-\nabla \cdot p\nabla u(x,y) + qu(x,y) = f(x,y), \quad x,y\; \Omega \subset \mathbb{R}^2 \tag{4.1}$$

subject to boundary condition

$$-(\hat{n} \cdot p\nabla u) = c(u-d) - e \;\text{ on }\; \partial\Omega \tag{4.2}$$

where $\hat{n}$ is an outward pointing unit vector, $\nabla := \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$, $p(x) > 0$, $q(x) \geqslant 0$ and $f(x,y)$ are given functions with the following given constants c, d and e. Furthermore, Ω is a domain.

4.3 Variational formulation

Green's formula

Let $\bar{F}$ be a vector field on the domain Ω with boundary $\partial\Omega$ then from the divergence theorem we have that

$$\int_{\Omega} \nabla \cdot \bar{F} d\mathbf{x} = \int_{\partial\Omega} \hat{n} \cdot \bar{F} ds \tag{4.3}$$

where $\hat{n}$ is an outward pointing unit vector and $\mathbf{x} = (x, y) \in \mathbb{R}$. Setting $\bar{F} = v\nabla u$ we obtain

$$\int_{\Omega} \nabla \cdot v\nabla u d\mathbf{x} = \int_{\partial\Omega} \hat{n} \cdot v\nabla u ds \qquad \text{where } \mathbf{x} = (x, y) \in \mathbb{R} \tag{4.4}$$

Expanding the left hand side using product rule we get

$$\int_{\Omega} (v\nabla \cdot \nabla u + \nabla u \cdot \nabla v) d\mathbf{x} = \int_{\partial\Omega} \hat{n} \cdot v\nabla u ds \tag{4.5}$$

Rearranging (4.5) gives Green's formula

$$\int_{\Omega} v\nabla \cdot \nabla u d\mathbf{x} = \int_{\partial\Omega} \hat{n} \cdot v\nabla u ds - \int_{\Omega} \nabla u \cdot \nabla v d\mathbf{x} \tag{4.6}$$

Let

$$V := \{v : \|v\|_{L^2(\Omega)} + \|\nabla v\|_{L^2(\Omega)} < \infty\}$$

be the space of square integrable functions and whose first derivatives are also square integrable in the domain Ω.
Multiplying equation (4.1) by a test function $v \in V$ and integrating we get

$$\begin{aligned}\int_{\Omega} fv d\mathbf{x} &= -\int_{\Omega} v\nabla \cdot p\nabla u d\mathbf{x} + \int_{\Omega} quv d\mathbf{x} \\ &= \int_{\Omega} p\nabla u \cdot \nabla v d\mathbf{x} + \int_{\Omega} quv d\mathbf{x} + \int_{\partial\Omega} v(-\hat{n} \cdot p\nabla u) ds \quad \text{(Green's formula)}\end{aligned}$$

Substituting the boundary conditions and simplifying we obtain

$$\int_{\Omega} fv d\mathbf{x} = \int_{\Omega} p\nabla u \cdot \nabla v d\mathbf{x} + \int_{\Omega} quv d\mathbf{x} + \int_{\partial\Omega} cuv ds - \int_{\partial\Omega} (cd + e) v ds$$

Hence the variational formulation: Find $u \in V$ such that

$$\int_{\Omega} (p\nabla u \cdot \nabla v + quv) d\mathbf{x} + \int_{\partial\Omega} cuv = \int_{\Omega} fv d\mathbf{x} + \int_{\partial\Omega} v(cd + e) ds \quad \forall v \in V. \tag{4.7}$$

4.4 Finite element approximation

Let $\mathcal{K}$ be a triangulation which consists of n_t triangles and n_p nodes. Associated with each node $N_i := (x_i, y_i)$ is the hat function $\varphi_i(x, y)$. Let

$$V_h = \{v \in V | v \textit{ is continous piecewise linear on } \mathcal{K}\}$$

be the space of continous piecewise linear functions on $\mathcal{K}$. The finite element method consists of finding $u_h \in V_h$:

$$A(u_h, v) = l(v), \quad \forall v \in V_h \tag{4.8}$$

where

$$A(u_h, v) = \int_\Omega (p\nabla u_h \cdot \nabla v + q u_h v) d\mathbf{x} + \int_{\partial\Omega} c u_h v ds$$

and

$$l(v) = \int_\Omega f v d\mathbf{x} + \int_{\partial\Omega} (cd + e) v ds$$

4.5 Existence and uniqueness of weak solution

In this section, we prove the existence and uniquenes of the weak solution u_h. We make use of the bilinear form

$$A(u, v) := (p\nabla u, \nabla v) + (qu, v) = \int_\Omega (p\nabla u \cdot \nabla v + quv) d\mathbf{x},$$

linear form

$$l(v) := (f, v) = \int_\Omega f v d\mathbf{x}$$

and Hilbert space $V = H_0^1(\Omega)$ with

$$H_0^1(\Omega) = \{w : \Omega \to \mathbb{R}^2 | \ \|w\|_V^2 := \|\nabla w\|^2 + \|w\|^2 < \infty\} \tag{4.9}$$

The linearity of $l(\cdot)$ follows from the fact that integration and defferentiation are linear. The continuity of $l(\cdot)$ follows in a similar manner to the linear form $l(\cdot)$ for the poisson equation in $1-d$. What is left is to prove the continuity and coercivity of $A(\cdot, \cdot)$.

The continuity of the bilinear form follows from the triangle inequality as shown below

$$\begin{aligned}
|A(u, v)| &:= |(p\nabla u, \nabla v) + (qu, v)| \\
&\leqslant \|p\nabla u\|\|\nabla v\| + \|qu\|\|v\| && \text{(C-S inequality)} \\
&\leqslant \|p\|_\infty \|\nabla u\|\|\nabla v\| + \bar{q}\|u\|\|v\| && (\bar{q} := \max_{x\in\Omega} q(x)) \\
&\leqslant C(\|\nabla u\|\|\nabla v\| + \|u\|\|v\|) && (C = \max\{\|p\|_\infty, \bar{q}\}) \\
&\leqslant C(\|\nabla u\|^2 + \|u\|^2)^{\frac{1}{2}} (\|\nabla v\|^2 + \|v\|^2)^{\frac{1}{2}} && \text{(discrete C-S inequality)} \\
&= C\|u\|_V \|v\|_V.
\end{aligned}$$

The coercivity of A follows from

$$\begin{aligned}
A(u, u) &:= (p\nabla u, \nabla u) + (qu, u) \\
&\geqslant \bar{p}\|\nabla u\|^2 + \bar{q}\|u\|^2 && (\bar{p} := \min_{x\in\Omega} p(x) \text{ and } \underline{q} := \min_{x\in\Omega} q(x)) \\
&\geqslant C(\|\nabla u\|^2 + \|u\|^2) && \text{where } (C = \min\{\bar{p}, q\}) \\
&= C\|u\|_V^2
\end{aligned} \tag{4.10}$$

Invoking the Lax-Milgram lemma concludes the proof.

4.6 Derivation of a linear system

Equation (4.8) is equivalent to

$$\int_\Omega (p\nabla u_h \cdot \nabla\varphi_i + qu_h\varphi_i) + \int_{\partial\Omega} cu_h\varphi_i = \int_\Omega f\varphi_i + \int_{\partial\Omega} \varphi_i(cd+e) \tag{4.11}$$

for $i = 0, 1, 2, \ldots, n_i$. Since

$$u_h = \sum_{j=1}^{n_i} \xi_j\varphi_j \tag{4.12}$$

then upon interchanging diffentiation with summation and interchanging integration with summation equation (4.11) becomes

$$\sum_{j=1}^{n_i} \xi_j \int_\Omega (p\nabla\varphi_j\cdot \nabla\varphi_i + q\varphi_j\varphi_i) + \sum_{j=1}^{n_i} \xi_j \int_{\partial\Omega} c\varphi_j\varphi_i ds = \int_\Omega f\varphi_i + \int_{\partial\Omega} (cd+e)\varphi_i ds \tag{4.13}$$

with $j = 0, 1, 2 \ldots, n_i$. In matrix form, this is

$$(A + M + R)\xi = b + r \tag{4.14}$$

where

$$\begin{aligned} A_{ij} &= \int_\Omega p\nabla\varphi_j\cdot\nabla\varphi_i \\ M_{ij} &= \int_\Omega q\varphi_j\varphi_i \\ R_{ij} &= \int_{\partial\Omega} c\varphi_j\varphi_i ds \quad (c \text{ is a constant}). \\ b_i &= \int_\Omega f\varphi_i \end{aligned}$$

and

$$r_i = \int_{\partial\Omega} (cd+e)\varphi_i ds$$

4.7 Assembly

In this section we assemble matrices A, M, R, b and r in equation (4.14). Let K be a triangle in the mesh $\mathcal{K}$ on Ω. See Figure 4.1. Let $|K|$ be the area of K and let $N_i(x_i, y_i)$, $i = 1, 2, 3$ denote the nodes of K.
The following quadrature rules can be used to compute the integrals over each triangle K.

1. The centre of gravity rule

$$\int_K f dx \approx f\left(\frac{N_1 + N_2 + N_3}{3}\right)|K| \tag{4.15}$$

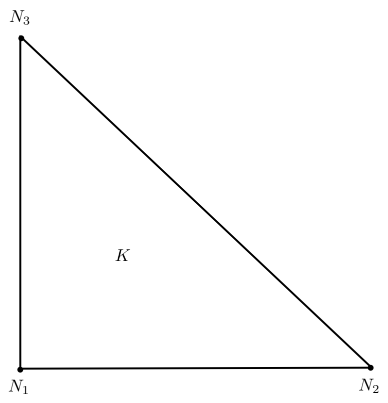


Figure 4.1: Triangle K with 3 nodes N_1, N_2 and N_3.

2. The corner quadrature formula

$$\int_K f dx \approx \sum_{i=1}^{3} f(N_i)\frac{|K|}{3} \tag{4.16}$$

3. 2-dimensional mid-point rule

$$\int_K f dx \approx \sum_{1\leq i,j\leq 3}^{3} f\left(\frac{N_i+N_j}{2}\right)\frac{|K|}{3} \tag{4.17}$$

4.7.1 Assembly of the stiffness matrix

Since

$$\int_\Omega p\nabla\varphi_i\cdot\nabla\varphi_j = \sum_{K\in\mathcal{K}}\int_K p\nabla\varphi_i\cdot\nabla\varphi_j,\ i=1,2,3. \tag{4.18}$$

The entries of the 3×3 local element stiffness matrix A corresponding to a triangle K with nodes $N_i = (x_1^{(i)}, x_2^{(i)}), i = 1,2,3$ are given by

$$A_{ij}^K = \int_K p\nabla\varphi_i\cdot\nabla\varphi_j,\ i=1,2,3. \tag{4.19}$$

Since the hat function φ_i associated with each node N_i on triangle K is linear, it can be written as

$$\varphi_i = a_i + b_i x_1 + c_i x_2$$

where $a_i, b_i, c_i \in \mathbb{R}$. Hence, the gradient of φ_i

$$\nabla\varphi_i = b_i + c_i = [b_i, c_i]^T \tag{4.20}$$

is a constant vector. Substituting (4.20) into (4.19) and computing it using the centre of gravity rule we get

$$A_{ij}^K = \int_K p(b_i b_j + c_i c_j) \approx (b_i b_j + c_i c_j) p \left(\frac{N_1 + N_2 + N_3}{3} \right) |K| \tag{4.21}$$

where $|K|$ is the area of triangle K. The necessary steps for assembling the *stiffness matrix* A are summarized by Algorithm 6.

Algorithm 6: Assembling stiffness matrix

```
1  Let $P$ be point matrix and $T$ be connectivity matrix for the mesh $\mathcal{K}$ with
     $n_p$ nodes and $n_t$ triangles.
2  Initialise the $n_p \times n_p$ stiffness matrix $A$ as a zero matrix.
3  for $K = 1, 2, \cdots, n_t$ do
4      Compute the gradients $\nabla\varphi_i = (b_i, c_i);\ i = 1, 2, 3$ on K.
5      Compute local element stiffness matrix
           $$A^K = \bar{p}|K| \begin{pmatrix} b_1^2 + c_1^2 & b_1 b_2 + c_1 c_2 & b_1 b_3 + c_1 c_3 \\ b_2 b_1 + c_2 c_1 & b_2^2 + c_2^2 & b_2 b_3 + c_2 c_3 \\ b_3 b_1 + c_3 c_1 & b_3 b_2 + c_3 c_2 & b_3^2 + c_3^2 \end{pmatrix}$$
         where $\bar{p} = p\left(\frac{N_1+N_2+N_3}{3}\right)$.
6      Set up the local-to-global mapping, loc2glb $= [r, s, t]$.
7      for $i = 1, 2, 3$ do
8          for $j = 1, 2, 3$ do
9              $A_{loc2glb_i, loc2glb_j} = A_{loc2glb_i, loc2glb_j} + A_{ij}^K$
10         end
11     end
12 end
```

4.7.2 Assembly of the mass matrix

Since

$$M_{ij} = \int_\Omega q\varphi_i\varphi_j = \sum_{K \in \mathcal{K}} \int_K q\varphi_i\varphi_j = \sum_{K \in \mathcal{K}} M_{ij}^K \tag{4.22}$$

and on triangle K, $\varphi_l \neq 0$ only when l is one of the nodes r, s, t, then

$$M_{ij}^K = \begin{cases} \int_K q\varphi_i\varphi_j \neq 0, & i, j = r, s, t \\ 0, & \text{otherwise.} \end{cases}$$

To assemble M, we need

- Local-to-global mapping: $\{r, s, t\} \mapsto \{1, 2, 3\}$.
- Integration formula:

$$\int_K \varphi_1^m \varphi_2^n \varphi_3^p = \frac{2m!n!p!}{(m + n + p + 2)!} A_K \tag{4.23}$$

  where A_K is the area of triangle K, and $m, n, p \in \{0, 1, 2, \cdots\}$.

Substituting $m = n = 1$ and $p = 0$ on the formula 4.23, we obtain

$$\int_K \varphi_1\varphi_2 dx = \begin{cases} \frac{2}{4!}A_K = \frac{1}{12}A_K, & \text{when } \varphi_1 \neq \varphi_2 \\ \frac{2(2!)}{4!}A_K = \frac{2}{12}A_K, & \text{when } \varphi_1 = \varphi_2 \end{cases}$$

Hence, the above equations suggest that entries of the local mass matrix M^K are given by

$$M_{ij}^K = \int_K q\varphi_i\varphi_j = \frac{q}{12}(1+\delta_{ij})A_K \quad \text{for} \quad i,j = 1,2,3 \tag{4.24}$$

where $\delta_{ij} = 0$ if $i = j$ otherwise 0. In matrix form

$$M^K = \frac{q}{12}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} A_K.$$

Algorithm 7: Assemble mass matrix

```
1  Let $P, T$ be point matrix, connectivity matrix for mesh $\mathcal{K}$ with $n_p$ nodes,
     $n_l$ elements.
2  Initialize mass matrix $M \in \mathbb{R}^{n_p \times n_p}$ as a zero matrix.
3  for $K = 1, 2, \cdots, n_l$ do
4      Form local element mass matrix $M^K = \frac{q}{12}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} A_K$ where $A_K$
         is the area of triangle $K$.
5      Define local-to-global mapping $loc2glb = [r, s, t]$.
6      for $i = 1, 2, 3$ do
7          for $i = 1, 2, 3$ do
8              $M_{loc2glb_i, loc2glb_j} = M_{loc2glb_i, loc2glb_j} + M_{ij}^K$
9          end
10     end
11 end
```

Algorithm 7 gives an outline of the assembly procedure for constructing mass matrix M.

4.7.3 Assembly of the load vector

By the linearity of the integral

$$b = \sum_{K \in \mathcal{K}} \underbrace{\int_K f\varphi_i}_{b^K}, \tag{4.25}$$

where

$$b_i^K = \int_K f\varphi_i, \ i = 1,2,3 \tag{4.26}$$

are the only non-zero terms of b^K. This follows from the fact that the only basis functions that are non-zero on triangle K are $\varphi_i; i = 1, 2, 3$. Using the corner quadrature rule gives

$$b_i^K = \int_K f\varphi_i \approx f(N_i)\frac{|K|}{3}, \; i = 1, 2, 3. \tag{4.27}$$

Therefore, the load vector is given by

$$b^K = \frac{|K|}{3}\begin{pmatrix} f(N_1) \\ f(N_2) \\ f(N_3) \end{pmatrix}$$

Algorithm 8 shows how the load vector b is assembled.

Algorithm 8: Assembling load vector

```
1 Let $P$ be point matrix and $T$ be connectivity matrix for the mesh $\mathcal{K}$ with
  $n_p$ nodes and $n_t$ triangles.
2 Initialise the $n_p \times 1$ load vector $b$ as a zero matrix.
3 for $K = 1, 2, \cdots, n_t$ do
4     Compute local element load vector
          $$b^K = \frac{|K|}{3}\begin{pmatrix} f(N_1) \\ f(N_2) \\ f(N_3) \end{pmatrix}$$
5     Set up the local-to-global mapping, loc2glb $= [r, s, t]$.
6     for $i = 1, 2, 3$ do
7         $b_{loc2glb_i} = b_{loc2glb_i} + b_i^K$
8     end
9 end
```

Algorithms 6,7 and 8 can be translated to their equivalent MATLAB codes respectively. The MATLAB codes were left out due to the fact that this chapter is beyond the scope of the study. However, this chapter can be extended in possible future work.

4.7.4 Assembly of the boundary matrices

Let E be the edge of triangle K which lies on the boundary $\partial\Omega$, and let N_i and N_j be two nodes at the ends of edge E. Let $\mathcal{E} = \{E\}$ be the set of all edges on $\partial\Omega$, then linearity of the integral implies that

$$R = \sum_{E\in\mathcal{E}} R^E \tag{4.28}$$

where

$$R^E = c\int_E \varphi_i\varphi_j ds.$$

Since on edge E the only non-zero basis functions are φ_i and φ_j, then the only non-zero entries of R^E are

$$R_{ij}^E = \int_E c\varphi_i\varphi_j \approx \frac{c|E|}{6}(1+\delta_{ij}), \quad i,j = 1,2 \tag{4.29}$$

where $|E|$ denotes the length of the edge E and nodes N_i and N_j have been tacitly replaced by N_1 and N_2 respectively.

Similarly, making use of the midpoint rule gives the only non-zero entries of the local boundary load vector

$$b_i^E = (cd+e)\int_E \varphi_i \approx (cd+e)\frac{|E|}{2}, \quad i = 1,2. \tag{4.30}$$

Just like in section 4.7.3 we intentionally skip including the code here. Even if we did not include our own code for the problem on this chapter, we can make use of MATLAB or Octave's PDE solvers. We demonstrate this in the following example.

4.7.5 Example

Consider the problem of finding solution $u(x,y)$ which satisfies the 2-d Poisson equation

$$-\Delta u(x,y) = f(x,y) \;\text{ on }\; \Omega \tag{4.31}$$

$$u_e = u_d = \frac{x^2+y^2}{4} \;\text{ on }\; \partial\Omega \tag{4.32}$$

where $\Delta = (\frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2})$, $\Omega = [0,1]^2$ is the unit square and $f(x,y) = 1$. This problem has exact solution $u_e = \frac{x^2+y^2}{4}$.

4.7.6 Solution

We made use of MATLAB's pdetool [13] and Python's FEniCS [8] to approximate the solution to equations (4.31) and (4.32). In Appendix A, we discuss how this solution was obtained using MATLAB's pdetool. In Appendix B we solve the same problem using Python's FEniCS.

4.8 Summary

In this chapter we used the FEM to numerically solve the RDE in the real plane. We used the Lax-Milgram lemma to prove the existence and uniqueness of the approximate solution. We used the MATLAB's pdetool and Python's FEniCS to find approximate solutions for a simple problem. The results basically look the same.

Chapter 5

Conclusion

In this study we used the FEM to solve boundary value problems for the reaction-diffusion equation on the real line and in the real plane. We used the Lax-Milgram lemma to prove the existence and uniqueness of the weak solutions. We used Octave to implement the FEM in $\mathbb{R}$ and produce results. In one example an exact solution existed and we used it to validate the finite element solution. In the real plane, we used MATLAB's pdetool and Python's FEniCS to solve the RDE. We did not make use of our own code in this case. Instead we used inbuilt code. Possible future work could involve using the FEM with adaptive mesh refinement in $\mathbb{R}^2$. Furthermore free software libraries like deal.ii could be used.

Appendix A

Solution of the 2-d example problem using MATLAB

In this appendix, we present the code and the results generated by MATLAB's pdetool for the problem consisting of equations (4.31) - (4.32).

Model equation

We start by creating an empty model container for 1 partial differential equation (PDE) of the form

$$(1)\quad mu_{tt} + du_t - \nabla \cdot (c\nabla u) + au = f$$

To this end we make use of the inbuilt function `createpde`. This function returns a structure array with several empty fields for describing the problem and it is used as follows:

```
model = createpde(1);
```

We wish to solve equation (1) on the unit square $\Omega = [0, 1]^2$. To this end we add the unit square geometry to the model container as follows:

```
gd = [3  4  0  1  1  0  0  0  1  1]'; g = decsg(gd);
geometryFromEdges(model,g);
```

We create a mesh on Ω and add it to the model container:

```
generateMesh(model);
```

At this point, what does the `struct array` model contain?

```
model
```

```
model =
  PDEModel with properties:

           PDESystemSize: 1
         IsTimeDependent: 0
                Geometry: [1x1 AnalyticGeometry]
    EquationCoefficients: []
      BoundaryConditions: []
       InitialConditions: []
                    Mesh: [1x1 FEMesh]
           SolverOptions: [1x1 pde.PDESolverOptions]
```

Inspect the `Geometry` property in model:

```
geometry = model.Geometry
```

```
geometry =
  AnalyticGeometry with properties:

       NumCells: 0
       NumFaces: 1
       NumEdges: 4
    NumVertices: 4
```

We observe that Ω has 1 face, 4 edges and 4 vertices as expected.

Poisson’s equation

In this example we seek a solution to

$$(2) \quad -\Delta u = 1$$

This is a particular case of equation (1) with $m = d = 0, c = 1, a = 0$ and $f(x, y) = 1$. We specify these coefficients and add them to the model container.

```
specifyCoefficients(model,'m',0,'d',0,'c',1,'a',0,'f',1);
```

Boundary conditions

On the boundary $\partial\Omega$ of Ω we impose the dirichlet boundary condition

$$(3) \quad u = \frac{x^2 + y^2}{4} \text{ on } \partial\Omega$$

In MATLAB this is achieved by doing the following:

```
applyBoundaryCondition(model,'dirichlet','Edge',1:model.Geometry.NumEdges,'u',@rfun);
```

which means that we are imposing the zero dirichlet boundary condition on all edges of $\partial\Omega$.

Inspect `model`:

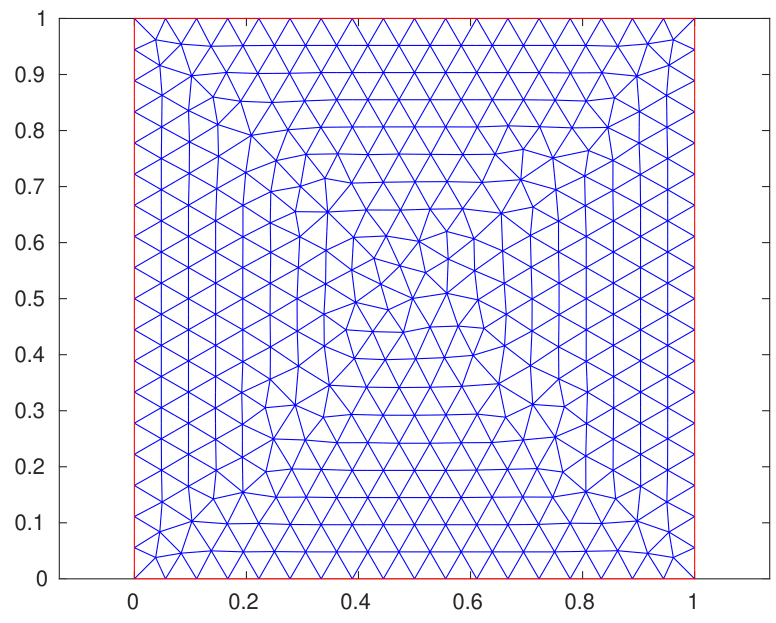
```
model
```

```
model =
  PDEModel with properties:

                 PDESystemSize: 1
               IsTimeDependent: 0
                      Geometry: [1x1 AnalyticGeometry]
          EquationCoefficients: [1x1 CoefficientAssignmentRecords]
            BoundaryConditions: [1x1 BoundaryConditionRecords]
             InitialConditions: []
                          Mesh: [1x1 FEMesh]
                 SolverOptions: [1x1 pde.PDESolverOptions]
```

and plot mesh

```
pdeplot(model.Mesh)
```



Solve PDE and plot solution

Finally we call `solvepde` to solve PDE (2) subject to boundary condition (3) based on the information in the model container then we plot the solution with the mesh.

```
results = solvepde(model); u = results.NodalSolution;
pdeplot(model,'XYData',u,'ZData',u,'Mesh','on')
xlabel('x'); ylabel('y'); zlabel('u');
title('-\Delta u = 1 in \Omega, u = (x ^ 2 + y ^ 2)/4 on \partial\Omega')
```

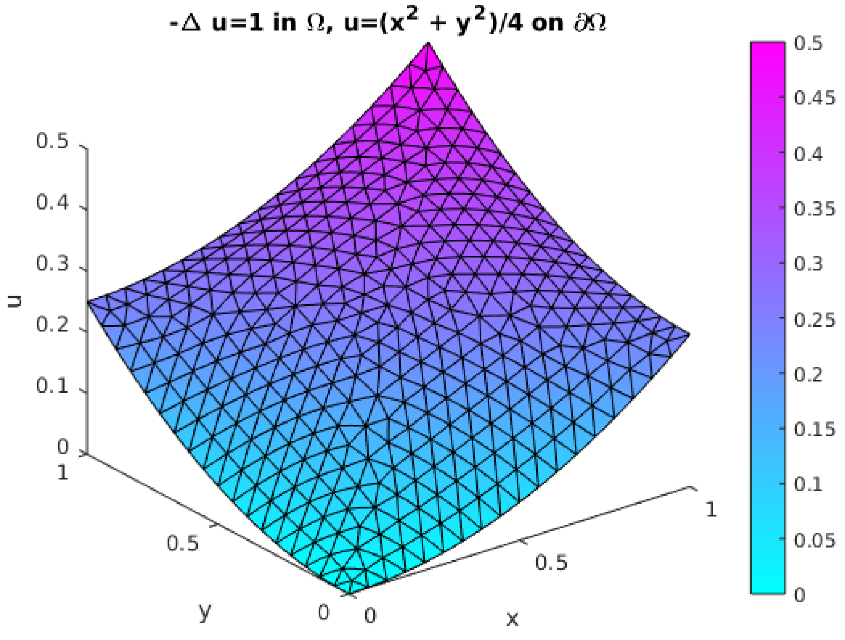


Figure A.1 Finite element solution.

Matlab function for non-constant boundary condition

```
function r = rfun(location,~)
    x = location.x; y = location.y;  %  (x,y)
    r = (x. ^ 2 +  y. ^ 2)/4;  %  u = r on boundary
end
```

Shown in Figure A.1 is the finite element solution to the problem consisting of equations (2) and (3).

Appendix B

Solution of the 2-d example problem using Python

In this appendix we present finite element solution of the equations (4.31) and (4.32) obtained by making use of Python's FEniCS PDE solver. Shown in Listing B.1 is the Python program that was used to solve problem (4.31) with boundary condition (4.32).

Listing B.1: Python code generating solution to the problem (4.31).

```
"""
Solve bounday value problem:
    −del u=f in Omega
    u=u_D on boundary_of_Omega
where:
    f=1 and
    u_D=(x*x + y*y)/4
"""
from fenics import * # import key classes from FEniCS library
"""
 create mesh using the inbuilt UnitSquare class
"""
mesh=UnitSquareMesh(13,13)
"""
Define function space of (continuous) piecewise (Lagrange)
polynomials of degree 1 on the mesh of triangles
"""
V=FunctionSpace(mesh,'Lagrange',1)
"""
Set u=u_D:=(x*x + y*y)/4 (polynomial in x and y of degree 3)
on the boundary
"""
u_D=Expression('(x[0]*x[0]+x[1]*x[1])/4',degree=2)
def boundary(x,on_boundary):
    return on_boundary # Is a point x on the boundary or not?
bc=DirichletBC(V,u_D,boundary) # set u=u_D on boundary
# define (abstract) variational problem
```

```
u=TrialFunction(V) # u is a trial function in function space V
v=TestFunction(V) # v is similarly in V
f=Expression('1.0',degree=0) # RHS of Poisson's equation=-y
a=dot(grad(u),grad(v))*dx # define bilinear form a(u,v)
L=f*v*dx # define linear form L(v)
# compute solution
u=Function(V) # redefine u as the finite element solution
solve(a==L,u,bc) # solve variational problem for u with given bc
"""
 open file for exporting solution
"""
vtkfile=File("2-D_example_latex_file/eg_2_solution.pvd")
vtkfile << u # write solution u to file
l2_norm_error=errornorm(u_D,u,norm_type="l2") # ||u-uh|| L2 norm
print('L2 norm error = ',l2_norm_error)
```

The solution that was generated by the program in Listing B.1 was exported to the software Paraview for plotting the solution. See Figure B.1. Furthermore, this program also calculate the L^2- norm error $\|u - u_h\|_{L^2(\Omega)}$ to provide a comparison of the exact solution $u_e = \frac{x^2+y^2}{4}$ with the finite element solution u_h shown in Figure B.1. The L^2- norm error obtained was 0.0817069256190016.

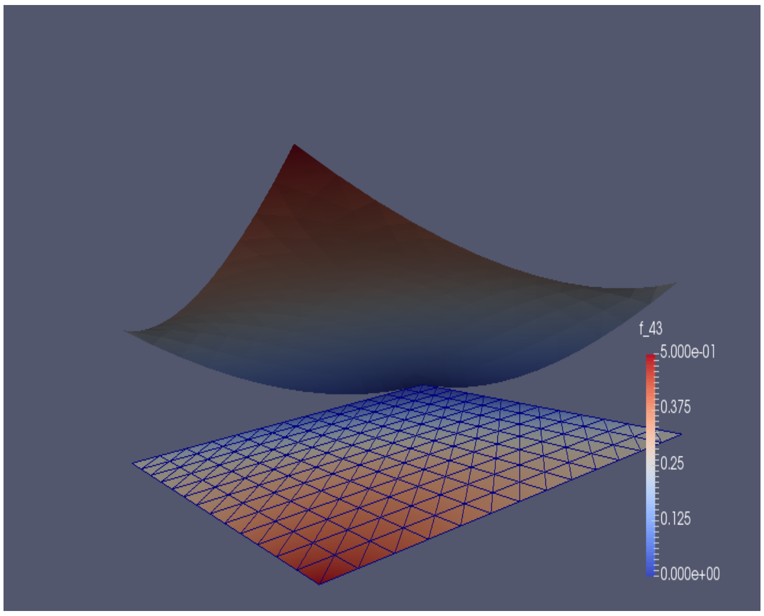


Figure B.1: The solution of the problem (4.31) and (4.32) using Python's PDE solver.

Bibliography

[1] J Alvarez-Ramirez, F J Valdes-Parada, J Alvarez, J A Ochoa-Tapia, Non-standard finite difference scheme for reaction-diffusion equations in curvilinear coordinates, Computer and Chemical Engineering, Vol 3, Issue 1, 2009, pp 277-286.

[2] G Akram, M Sadaf, Application of homotopy analysis method to the solution of ninth order boundary value problem in AFTI-F16 fighter, Journal of the Association of Arab Universities for Basic and Applied Science, 2016, http://dx.doi.org/10.1016/j.jaubus.2016.08.002, pp 1-7.

[3] M Ashyraliyev, J G Blom, J G Verwer, On the numerical solution of diffusion-reaction equations with singular source terms, Journal of Computational and Applied Mathematics, Vol 216, Issue 1, 2008, pp 20-38.

[4] A H Bhrawy, E H Doha, M A Abdelkawy, R A Van Gorder, Jacobi-Guass-Lobatto collocation method for solving nonlinear reaction-diffusion equations subject to Dirichlet boundary conditions, Applied Mathematical Modelling, Vol 40, 2016, pp 1703-1716.

[5] T U Chaudhari, D M Patel, Finite element solution of Poisson's equation in a homogeneous medium, International Research Journal of Engineering and Technology (IRJET), Vol 02, Issue 9, 2015, pp 674-679.

[6] A Ghorbani, M Bakherad, A variational iteration method for solving nonlinear Lane-Emden Problems, New Astronomy, Vol 54, 2017, pp 1-6.

[7] D Kumar, J Singh, S Kumar, Numerical computation of Klein-Gordon equations arising in quantum field theory by using homotopy analysis transform method, Alexandria Engineering Journal, Vol 53, Issue 2, 2014, pp 469-474.

[8] H P Langtangen, A Logg, Solving PDEs in Python, Simula SpringerBriefs on Computing, Vol 03, 2016.

[9] A Madzvamuse, A H W Chung, Fully implicit time-stepping schemes and non-linear solvers for systems of reaction-diffusion equations, Applied Mathematics and Computation, Vol 244, 2014, pp 361-374.

[10] G Makanda, Numerical study of convective fluid flow in porous and non-porous media, PhD dissertation, University of KwaZulu Natal, 2015.

[11] B Muatjetjela, C M Khalique, Exact solutions of the generalised Lane-Emden equations of the first and second kind, Journal of Physics, Vol 77, No 3, 2011, pp 545-554.

[12] D Priimak, Finite difference numerical method for superlattice Boltzmann transport equation and case comparison of CPU(C) and GPU(CUDA) implementations, Journal of Computational Physics, Vol 278, 2014, pp 182-192.

[13] A Quarteroni, F Saleri, P Gervasio, Scientific Computing with MATLAB and Octave, Springer, 2014.

[14] R N Rao, P P Chakravarthy, A finite difference method for singularly perturbed differential difference equations with layer and oscillatory behaviour, Applied Mathematical Modelling, Vol 37, Issue 8, 2013, pp 5743-5755.

[15] D Shanthi, V Ananthaswamy, L Rajendran, Analysis of non-linear reaction-diffusion processes with Michaelis-Menten kinetics by a new Homotopy perturbation method, Natural Science, 5(9), 2013,pp 1034-1046.

[16] W Sikander, U Khan, N Ahmed, Optimal variation iteration method for nonlinear problems, Journal of the Association of Arab Universities for Basic and Applied Science, 2016. http://dx.doi.org/10.1016/j.jaubus.2016.09.004, pp 1-7.

[17] Y M Wang, H B Zhang, Higher-order compact finite difference method for system of reaction-diffusion equations, Journal of Computational and Applied Mathematics, Vol 233, 2009, pp 502-518.