Numerical Solutions of the Diffusion Equation Using the Finite Element Method

C6.4 Finite Element Methods for PDEs

Jake Bowhay

1 Introduction

The finite element method is ubiquitous in computational science and engineering because it can handle complex geometries, solve a wide range of equations, and preserve the original properties of the problem [12]. Due to the complexity associated with this flexibility, sophisticated software packages such as FEniCS [2] and Firedrake [13] are often used, as these tools are designed to make the finite element method more accessible to general scientists and engineers. Furthermore, the aforementioned tools provide additional features, such as access to sophisticated, performant, and highly scalable linear and non-linear solvers, such as those provided by PETSc [1]. However, the abstraction provided by such tools also obscures some of the basic operations required in the finite element method from the user, such as the assembly of the stiffness matrix. Hence, for pedagogical reasons, in this report we will detail solving from scratch a simple model problem, the diffusion equation, on an arbitrary, two-dimensional domain. The diffusion equation has widespread applications throughout engineering and the physical sciences and so is a natural choice as an illustrative example to demonstrate solving a transient problem using the finite element method.

In this report, we convert the strong form of the diffusion equation problem into a semi-discretised variational form and show its well-posedness. Next, we introduce a Galerkin approximation leading to a method of lines scheme. The implicit Euler method is then used to discretise the time derivative, for which we present a stability and consistency result. Then we show how the finite element method is used to construct the function space in which the solution lies. Furthermore, we explain how the mass and stiffness matrices and the forcing vector are computed and used in the scheme. Finally, we verify the convergence of the numerical scheme on a triangular, square and pentagonal domain by comparing it against known solutions generated using the method of manufactured solutions.

2 Problem Statement

The diffusion equation with associated boundary and initial conditions are given in strong form by

$$\begin{cases} \partial_t u - \kappa \Delta u = f(x,y,t) & \text{in} \quad \Omega \times [0,T], \\ u = 0 & \text{on} \quad \partial\Omega \times [0,T], \\ u(x,y,0) = u_0(x,y) & \text{in} \quad \Omega, \end{cases} \tag{1}$$

where $\Omega \subset \mathbb{R}^2$ is a bounded, Lipschitz continuous domain with boundary $\partial\Omega$, κ is the diffusivity constant, and $T > 0$ is the final time to solve the problem to. For simplicity, we only consider homogeneous Dirichlet boundary conditions. Here, for example, the problem (1) could model heat transfer in a material or the diffusion of a chemical.

3 Mathematical Background

In this section, we introduce some function spaces and concepts required for the analysis of (1). Instead of considering the solution u to be a function of both space and time, it is helpful, for the construction of the variational form and associated well-posedness proof, to consider u as a function of t with values in a Banach space V. The elements of V are then functions which only depend on space. Hence, following [10, 9, 11], we define u by the mapping

$$u : [0, T] \to V, \tag{2}$$

such that

$$u(x, y, t) \equiv [u(t)](x, y), \tag{3}$$

where $t \in [0, T]$ and (x, y) is in the domain of the space V.

Next, we proceed to introduce the following function spaces required to construct the variational form of (1).

Definition 1. *Following [10, 11], the Banach space $\mathcal{C}^j([0, T]; V)$, $j \geq 0$, is the space of V-valued functions that are $\mathcal{C}^j$ with respect to t. We define the associated norm as*

$$\|u\|_{\mathcal{C}^j([0,T];V)} = \sup_{t\in[0,T]} \sum_{l=0}^{j} \|\partial_t^l u(t)\|_V, \tag{4}$$

where $\partial_t^l u$ denotes the time derivative of order l of u.

Definition 2. *Following [10, 11], the Banach space $L^p((0, T); V)$, $1 \leq p \leq \infty$ is the space of V-valued functions whose norm in V is in $L^p((0, T))$. We define the associated norm as*

$$\|u\|_{L^p((0,T);V)} = \begin{cases} \left(\int_0^T \|u(t)\|_V^p \,\mathrm{d}t\right)^{\frac{1}{p}} & \text{if } 1 \leq p < \infty, \\ \operatorname{ess\,sup}_{t\in(0,T)} \|u(t)\|_V & \text{if } p = \infty. \end{cases} \tag{5}$$

Definition 3. *Following [10], let* $p_1, p_2 \in (1, \infty)$ *and* $B_0 \subset B_1$ *be two reflective Banach spaces with continuous embeddings. We define the following Banach space*

$$\mathcal{W}(B_0, B_1) = \{v : (0, T) \to B_0;\ v \in L^{p_1}((0, T); B_0);\ \partial_t v \in L^{p_2}((0, T); B_1)\}, \tag{6}$$

with the associated norm

$$\|u\|_{\mathcal{W}(B_0,B_1)} = \|u\|_{L^{p_1}((0,T);B_0)} + \|\partial_t u\|_{L^{p_2}((0,T);B_1)}. \tag{7}$$

Here ∂_t *denotes the time-derivative of* u *in the distributional sense.*

4 Semi-Discretised Variational Form

In this section, we seek to find a variational form of the problem (1). We note that it is possible to discretise either in time or space first. Following Ern and Guermond [10, 9], we take the latter approach, commonly called a method of lines discretisation, to first find the so-called semi-discretised system. We will then discretise in time in Section 7.

To find the variational form of (1), we assume $f \in L^2((0, T), H^{-1}(\Omega))$ and $u_0 \in L^2(\Omega)$. Then we multiply (1) by a test function $v \in V$, where V is a space of sufficiently regular functions, and integrate over Ω, giving

$$\int_\Omega \partial_t uv \,\mathrm{d}\mathbf{x} - \kappa \int_\Omega \Delta uv \,\mathrm{d}\mathbf{x} = \int_\Omega fv \,\mathrm{d}\mathbf{x}. \tag{8}$$

Then we apply Green's first identity to shift one derivative of the Laplace operator onto the test function, giving

$$\int_\Omega \partial_t uv \,\mathrm{d}\mathbf{x} + \kappa \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x} - \kappa \int_{\partial\Omega} v(\nabla u \cdot \mathbf{n}) \,\mathrm{d}\mathbf{s} = \int_\Omega fv \,\mathrm{d}\mathbf{x}, \tag{9}$$

where $\mathbf{n}$ is the outward pointing unit normal to the surface element $\mathrm{d}\mathbf{s}$. We choose v such that it vanishes on the boundary and hence we see that the correct function space V to choose test functions from is the Sobolev space $H_0^1(\Omega) = \{u \in H^1(\Omega) : u|_{\partial\Omega} = 0\}$. We denote the dual of this space by $H^{-1}(\Omega)$. Consequently, the surface integral vanishes giving

$$\int_\Omega \partial_t uv \,\mathrm{d}\mathbf{x} + \kappa \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x} = \int_\Omega fv \,\mathrm{d}\mathbf{x}. \tag{10}$$

We define the bilinear form

$$a(t; u, v) = \kappa \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x}. \tag{11}$$

The bilinearity of $a(t; u, v)$ in u and v trivially follows from the linearity of integration and the gradient operator. The variational problem is then given by:

$$\begin{cases} \text{Find } u \in \mathcal{W}(H_0^1(\Omega), H^{-1}(\Omega)) \text{ such that, a.e. } t, \ \forall v \in H_0^1(\Omega), \\ \langle \partial_t u, v \rangle_{H^{-1}, H_0^1} + a(t; u, v) = \langle f, v \rangle_{H^{-1}, H_0^1}, \\ u(0) = u_0. \end{cases} \tag{12}$$

Remark 4. *$\langle \cdot, \cdot \rangle_{H^{-1}, H_0^1}$ denotes the duality pair of $H^{-1}(\Omega)$ and $H_0^1(\Omega)$. As noted in [10, 9], under the assumption that $\partial_t u, f \in L^2((0, T), H^{-1}, (\Omega))$, the duality pair is realised through the L^2 inner product*

$$(u, v)_{L^2(\Omega)} = \int_\Omega uv \, \mathrm{d}\mathbf{x}. \tag{13}$$

5 Well-Posedness of the Semi-Discretised Variational Form

Next, we prove a series of results to show that (12) has a unique solution. As a prerequisite, we first introduce the following important inequality which is used to show the bilinear form is coercive. Then we then proceed to show these results lead to the well-posedness of (12).

Theorem 5 (Poincaré–Friedrichs inequality [12])**.** *Let Ω be a bounded, Lipschitz domain and with a closed boundary $\partial\Omega$ that has nonzero measure. Then there exists a constant $K < \infty$, known as the Poincaré constant, that only depends on Ω and $\partial\Omega$, such that*

$$\int_\Omega u^2 \, \mathrm{d}\mathbf{x} \leq K \int_\Omega |\nabla u|^2 \, \mathrm{d}\mathbf{x} \quad \forall u \in H_0^1(\Omega). \tag{14}$$

Theorem 6 (Measurability of a)**.** *The function $t \mapsto a(t; u, v)$ is measurable for all $u, v \in H_0^1(\Omega)$.*

Proof. The bilinear form $a(t; u, v)$ is continuous with respect to time and therefore is measurable [11, 9]. □

Theorem 7 (Continuity of a)**.** *There exists an $M < \infty$ such that*

$$|a(t; u, v)| \leq M \|u\|_{H^1(\Omega)} \|v\|_{H^1(\Omega)} \quad \text{for a.e. } t \in [0, T], \ \forall v, u \in H_0^1(\Omega). \tag{15}$$

Proof.

$$\begin{aligned}|a(t;u,v)| &= \kappa \left|(\nabla u, \nabla v)_{L^2(\Omega)}\right| \\ &\leq \kappa \|\nabla u\|_{L^2(\Omega)} \|\nabla v\|_{L^2(\Omega)} \\ &\leq \kappa \left(\|u\|^2_{L^2(\Omega)} + \|\nabla u\|^2_{L^2(\Omega)}\right)^{\frac{1}{2}} \left(\|v\|^2_{L^2(\Omega)} + \|\nabla v\|^2_{L^2(\Omega)}\right)^{\frac{1}{2}} \\ &= \kappa \|u\|_{H^1(\Omega)} \|v\|_{H^1(\Omega)}.\end{aligned}$$

Hence, the continuity of a is satisfied with $M = \kappa$. □

Remark 8. *In the case κ is not constant and is instead defined as $\kappa : \Omega \times [0,T] \to \mathbb{R}$, carrying out the same proof mutatis mutandis we would instead find $M = \|\kappa\|_{L^\infty(\Omega\times[0,T])}$.*

Theorem 9 (Coercivity of a)**.** *There exists an $\alpha > 0$ such that*

$$a(t;u,u) \geq \alpha \|u\|^2_{H^1(\Omega)} \quad \textit{for a.e. } t \in [0,T],\ \forall u \in H^1_0(\Omega). \tag{16}$$

Proof. From the Poincaré–Friedrichs inequality (Theorem 5) we have

$$\begin{aligned}K a(t;u,u) &= \kappa K \int_\Omega |\nabla u|^2 \,\mathrm{d}\mathbf{x} \\ &\geq \kappa \int_\Omega u^2 \,\mathrm{d}\mathbf{x}.\end{aligned}$$

Then, adding $a(t;u,u)$ to both sides we get

$$\begin{aligned}(K+1)a(t;u,u) &\geq \kappa \left(\int_\Omega u^2 \,\mathrm{d}\mathbf{x} + \int_\Omega |\nabla u|^2 \,\mathrm{d}\mathbf{x}\right) \\ &= \kappa \|u\|^2_{H^1(\Omega)}.\end{aligned}$$

Finally, rearranging gives the desired result

$$a(t;u,u) \geq \frac{\kappa}{K+1} \|u\|^2_{H^1(\Omega)}.$$

Hence, the coercivity of a is satisfied with $\alpha = \kappa/K+1$, where K is the Poincaré constant. □

Remark 10. *Again, if κ was instead defined as $\kappa : \Omega \times [0,T] \to \mathbb{R}$ and is bounded bellow such that $\kappa \geq \tilde{\kappa}$, we would have $\alpha = \tilde{\kappa}/K+1$.*

Theorem 11 (J.L. Lions [10, 9])**.** *As a result of Theorem 6, Theorem 7, and Theorem 9,* (12) *has a unique solution.*

Proof. See [10] or [9]. □

6 Galerkin Approximation

Whilst we have shown the variational problem (12) to be well-posed, the infinite-dimensional test space $V = H_0^1(\Omega)$ is not practical to compute with. Instead, we seek a solution in the finite-dimensional space $V_h \subset V$. Hence, the Galerkin approximation to the problem is given by

$$\begin{cases} \text{Find } u_h \in \mathcal{C}^1([0,T];V_h) \text{ such that } \forall t \in [0,T],\ \forall v_h \in V_h, \\ (\partial_t u_h, v_h)_{L^2(\Omega)} + a(t; u_h, v_h) = (f, v_h)_{L^2(\Omega)}, \\ u_h(0) = u_h^0, \end{cases} \tag{17}$$

where $u_h^0 = \mathcal{I}_h u_0$ and, as defined in Definition 25, $\mathcal{I}_h$ is the global interpolation operator. The problem given by (17) is a finite system of linear ordinary differential equations (ODEs). Hence, the existence and uniqueness of the solution follows from the Picard–Lindelöf theorem [10, 18, 4].

Definition 12 (Lipschitz Map [4])**.** *Let V be a Banach space and $F : V \to V$ be a Lipschitz map, then there is a constant $L \in \mathbb{R}$ (the Lipschitz constant) such that for all $u, v \in V$*

$$\|Fu - Fv\| \leq L\|u - v\|. \tag{18}$$

Theorem 13 (Picard–Lindelöf [4])**.** *Given a Banach space V and a Lipschitz map $F : V \to V$, there exists a unique solution $u \in \mathcal{C}^1([0,\infty), V)$ to problem*

$$\begin{cases} u'(t) = Fu(t) \quad t \geq 0, \\ u(0) = u_0, \end{cases} \tag{19}$$

given initial data $u_0 \in V$.

Remark 14. *The Lipschitz continuity of* (17) *in u_h follows from the bilinearity and continuity (Theorem 7) of a. Hence there exists a unique solution $u_h \in \mathcal{C}^1([0,T], V_h)$ to problem* (17)*.*

Taking $\{\phi_1, \dots, \phi_N\}$ to be a basis for V_h, then for all $t \in [0,T]$ the solution $u_h(t) \in V_h$ can be written as

$$u_h(x,y,t) = \sum_{i=1}^{N} U_i(t)\phi_i. \tag{20}$$

We then define the stiffness matrix $\mathcal{A}(t) \in \mathbb{R}^{N \times N}$ by

$$\mathcal{A}_{ij}(t) = a(t; \phi_j, \phi_i), \tag{21}$$

the mass matrix $\mathcal{M} \in \mathbb{R}^{N \times N}$ by

$$\mathcal{M}_{ij} = (\phi_j, \phi_i)_{L^2(\Omega)}, \tag{22}$$

and the forcing vector $\mathcal{F}(t) \in \mathbb{R}^N$ by

$$\mathcal{F}_i(t) = (f(x,t), \phi_i)_{L^2(\Omega)}. \tag{23}$$

Since, for simplicity, we are considering κ to be constant, the stiffness matrix is also constant in time. Hence, from here onwards, we suppress its arguments. We can rewrite (17) in matrix-vector form as

$$\begin{cases} \mathcal{M}\partial_t \mathbf{u}(t) = -\mathcal{A}\mathbf{u} + \mathcal{F}(t), \quad t \in [0,T], \\ \mathbf{u}(0) = \mathbf{u}_0, \end{cases} \tag{24}$$

where $\mathbf{u}(t) \in \mathbb{R}^N$ is the solution vector and $\mathbf{u}_0 \in \mathbb{R}^N$ the vector respresentation of u_{0h}, both with respect to the basis of V_h.

7 Approximation in Time

Now that the problem has been transformed into a finite system of ODEs it remains to discretise the time derivative. For simplicity, we perform the time discretisation using the implicit Euler scheme. We note that, in practice, it would be typical to use a higher-order scheme than this, to achieve faster convergence in time.

To implement the implicit Euler scheme, we introduce a computational mesh in time. We let the integer $\bar{m} > 2$ be the number of temporal steps and hence the time step size is given by $\Delta t = {}^{T}/_{\bar{m}}$, where T is the final time to solve to. We denote the approximation of $\mathbf{u}(t_m)$ in the scheme by $\mathbf{U}^m$, where $t_m = m\Delta t$ for $m = 0, 1, \dots, \bar{m}$. The time derivative in the implicit Euler scheme is approximated by

$$\partial_t \mathbf{u}(t_m) \approx \frac{\mathbf{U}^{m+1} - \mathbf{U}^m}{\Delta t}. \tag{25}$$

Hence, we can rewrite (24) as a system of linear equations given by

$$\begin{cases} (\mathcal{M} + \Delta t \mathcal{A})\mathbf{U}^{m+1} = \mathcal{M}\mathbf{U}^m + \Delta t \mathcal{F}(t_{m+1}), \\ \mathbf{U}^0 = \mathbf{u}_0. \end{cases} \tag{26}$$

Here, we are assuming that the mesh used in the spatial discretisation is time-independent. In the implementation provided in Appendix D, we exploit the sparsity

of $\mathcal{M}$ and $\mathcal{A}$ by storing them as compressed sparse column arrays and by solving (26) using the interface to UMFPACK [7] provided by SciPy [21]. However, we note that, for larger problems, it would be necessary to switch to using an iterative method with a lower asymptotic computational complexity.

7.1 Stability of the Implicit Euler Scheme

Next, we seek to show that the scheme (26) is stable, by which we mean the solution of the scheme has a continuous dependence on the initial data and the forcing term f [19]. First, we introduce the following results which are required to show this.

Lemma 15 (Arthmetic-geometric inequality [10]). *For non-negative real numbers* $\{x_1, \dots, x_n\}$ *the following inequality holds*

$$(x_1 x_2 \dots x_n)^{\frac{1}{n}} \leq \frac{1}{n}(x_1 + \cdots + x_n). \tag{27}$$

Corollary 16. *When combined with the Cauchy-Schwarz inequality, Lemma 15 implies that, for all* $\gamma > 0$

$$(u, v) \leq \frac{\gamma}{2}\|u\|^2 + \frac{1}{2\gamma}\|v\|^2. \tag{28}$$

To show the scheme is stable, we note that the scheme can also be written in a variational form given by

$$\left(\frac{u_h^{m+1} - u_h^m}{\Delta t}, v_h\right)_{L^2(\Omega)} + a(t_{m+1}; u_h^{m+1}, v_h) = (f(x, y, t_{m+1}), v_h)_{L^2(\Omega)}, \tag{29}$$

where u_h^m denotes the approximation by the scheme to $u_h(t_m)$, rather than the matrix-vector notation previously used.

Theorem 17 (ℓ^2 Stability [9]). *For the solution* $u_{ht} = (u_h^1, u_h^2, \dots, u_h^{\bar{m}}) \in (V_h)^{\bar{m}}$ *and sequence of forcing terms* $f_t = (f(x, y, t_m))_{m \in \{0,1,\dots,\bar{m}\}}$*, which are a solution to the scheme* (29)*, we have the following stability result*

$$\alpha\|u_{ht}\|^2_{\ell^2([0,T],H^1(\Omega))} + \Delta t\|D_t^- u_{ht}\|^2_{\ell^2([0,T],L^2(\Omega))} + \|u_h^{\bar{m}}\|^2_{L^2(\Omega)} \leq \frac{1}{\alpha}\|f_t\|^2_{\ell^2([0,T],H^{-1})} + \|u_h^0\|^2_{L^2(\Omega)}, \tag{30}$$

where D_t^- *is the backwards difference operator and* α *is coercivity constant.*

Proof. Following [9], we test (29) with u_h^{m+1} and multiply through by Δt to get

$$\left(u_h^{m+1} - u_h^m, u_h^{m+1}\right)_{L^2(\Omega)} + \Delta t a(t; u_h^{m+1}, u_h^{m+1}) = \Delta t (f(x,y,t_{m+1}), u_h^{m+1})_{L^2(\Omega)}. \quad (31)$$

From the coercivity of $a(t_{m+1}; \cdot, \cdot)$ (Theorem 9) we have

$$\left(u_h^{m+1} - u_h^m, u_h^{m+1}\right)_{L^2(\Omega)} + \Delta t \alpha \|u_h^{m+1}\|^2_{H^1(\Omega)} \leq \Delta t (f(x,y,t_{m+1}), u_h^{m+1})_{L^2(\Omega)}. \quad (32)$$

Then, we note the following identity given in [9, 19]

$$\left(u_h^{m+1} - u_h^m, u_h^{m+1}\right)_{L^2(\Omega)} = \frac{1}{2}\|u_h^{m+1}\|^2_{L^2(\Omega)} - \frac{1}{2}\|u_h^m\|^2_{L^2(\Omega)} + \frac{1}{2}\|u_h^{m+1} - u_h^m\|^2_{L^2(\Omega)}, \quad (33)$$

which, when substituted into (32), gives

$$\|u_h^{m+1}\|^2_{L^2(\Omega)} - \|u_h^m\|^2_{L^2(\Omega)} + \|u_h^{m+1} - u_h^m\|^2_{L^2(\Omega)} + 2\Delta t \alpha \|u_h^{m+1}\|^2_{H^1(\Omega)} \\ \leq 2\Delta t (f(x,y,t_{m+1}), u_h^{m+1})_{L^2(\Omega)}. \quad (34)$$

We apply Corollary 16 to (34) to get

$$\|u_h^{m+1}\|^2_{L^2(\Omega)} - \|u_h^m\|^2_{L^2(\Omega)} + \|u_h^{m+1} - u_h^m\|^2_{L^2(\Omega)} + \Delta t \alpha \|u_h^{m+1}\|^2_{H^1(\Omega)} \\ \leq \frac{\Delta t}{\alpha}\|f(x,y,t_{m+1})\|^2_{L^2(\Omega)}. \quad (35)$$

Finally, summing over m, whilst noting that the first two terms result in a telescoping sum, and applying the definition of the backwards difference operator and the ℓ^2 norm, we get the required bound given in (30). □

7.2 Convergence of the Implicit Euler Scheme

Next, we analyse the error of the scheme, for which we introduce the timescale $\rho = {}^{2l_D^2}/{}_{K^2\alpha}$, where $l_D = \mathrm{diam}(\Omega)$, α is the coercivity constant, and K is the Poincaré constant.

Theorem 18 (ℓ^2 Error Estimate [9]). *Let $r \in [1,k]$, where k is the degree of finite element used in the construction of V_h, and let $u_t = (u(t_m))_{m\in\{0,1,\ldots,\bar{m}\}}$, then there exist constants $c_1, c_2 > 0$ such that*

$$\|u_t - u_{ht}\|_{\ell^2([0,T],H^1(\Omega))} \leq c_1 \Delta t \frac{\rho}{\ell_D}\|\partial_{tt}u\|_{L^2([0,T],L^2(\Omega))} \\ + c_2 h^r \left(\frac{M}{\alpha}|u_t|_{\ell^2([0,T],H^{r+1}(\Omega))} + \frac{\rho}{\ell_D}|\partial_t u|_{L^2([0,T],H^r(\Omega))} + \frac{1}{\sqrt{\alpha}}|u_0|_{H^r(\Omega)}\right), \quad (36)$$

where h is the maximum side length of a mesh element in the spatial mesh, where α is the coercivity constant, and M is continuity constant.

Proof. The proof of this result is beyond the scope of this report however it begins with the stability result shown in Theorem 17. See [9] for a complete proof. □

Remark 19. *From the previous theorem, we note that the scheme converges like* $\mathcal{O}(h^r + \Delta t)$ *in the* $\ell^2([0,T], H^1(\Omega))$ *norm.*

8 Implementation

In the previous sections, we conceived a scheme for solving (1) and showed the existence and uniqueness of the solution given by the scheme as well as the scheme's stability and convergence. In this section, we set out an explanation for how the scheme implemented in Appendix D works in practice.

8.1 Meshing and a Basis for V_h

In Section 6 we reduced the problem to a finite-dimensional system of equations by seeking a solution in $V_h \subset V$, called the Galerkin approximation. The finite element method provides a process for constructing the global space V_h out of local function spaces $\mathcal{P}$. The first step of the finite element process is subdividing the domain Ω into a mesh or triangulation, an example of which is shown in Figure 1. To generate the meshes used in this report we used the Python bindings to the Triangle software [17] provided by MeshPy [15].

Definition 20 (Mesh [10, 5])**.** *Let* $\Omega \subset \mathbb{R}^n$ *be a domain, then a mesh* $\mathcal{T}_h$ *of* Ω *is a union of* $N < \infty$ *sets* K_m*, called mesh elements, satisfying the following properties:*

i. $\bar{\Omega} = \bigcup_{m=1}^{N} K_m$.

ii. Each mesh element K_m *is closed and its interior* $\mathring{K}_m$ *is non-empty.*

iii. Each distinct pair of mesh elements K_m *and* K_n *satisfy* $\mathring{K}_m \cap \mathring{K}_n = \emptyset$.

iv. The boundary ∂K_m *of each mesh element* K_m *is Lipschitz continuous.*

With the domain triangulated into a mesh, we are then ready to introduce the definition of a finite element.

Definition 21 (Finite Element [10, 3, 5, 12])**.** *A finite element is a triple* $(K, \mathcal{P}, \mathcal{N})$ *where:*

i. K *is a mesh element, as per Definition 20.*

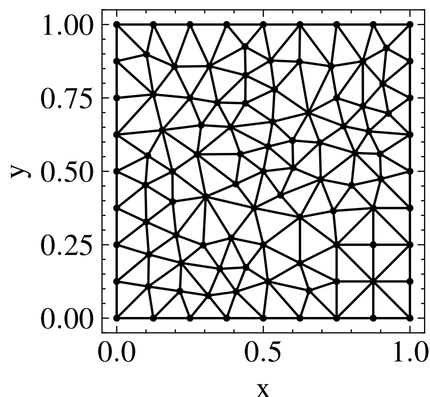

Figure 1: Example mesh of the domain $\Omega = [0,1] \times [0,1]$, produced using Triangle [17].

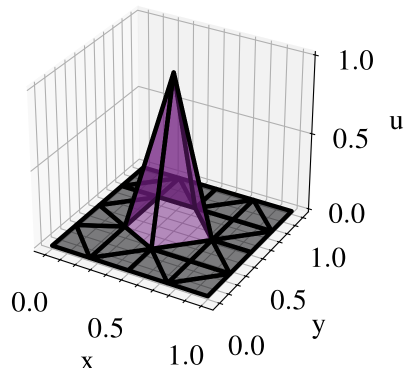

Figure 2: Example of a piecewise linear basis function, also known as a hat function (based on its appearance).

ii. $\mathcal{P}$ *is a finite-dimensional space of functions on* K.

iii. $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ *is a basis for the dual space of* $\mathcal{P}$.

In this report, for simplicity, we will exclusively use linear Lagrange elements, also known as the Courant triangle, an example of which is shown in Figure 2. We refer the interested reader to [3] or [10] for examples of other finite elements. For linear Lagrange elements, it is clear to see that if $u \in \mathcal{P}$ is zero at each vertex then u must be identically zero and, hence, the element is unisolvant.

Definition 22 (Linear Lagrange Elements on Triangles)**.** *The linear Lagrange element on a triangle* $(K, \mathcal{P}, \mathcal{N})$ *is defined as the following:*

i. K *is a triangular mesh element, as per Definition 20, with vertices* (x_1, y_1), (x_2, y_2), *and* (x_3, y_3).

ii. $\mathcal{P} = \operatorname{span}\{1, x, y\}$.

iii. The basis of $\mathcal{N}$ *is given by evaluation at the vertices of the mesh element. For* $u \in \mathcal{P}$, *the basis* $N_i : \mathcal{P} \to \mathbb{R}$ *is given by* $N_i(u) = u(x_i, y_i)$. *The value of* u *on the vertices is also known as the degrees of freedom.*

To express the solution in V_h it is helpful to introduce the concept of the nodal basis. This is because, for any $u^K \in \mathcal{P}$ and nodal basis $\{\phi_1^K, \phi_2^K, \phi_3^K\}$, it is possible to

express u^K as a linear combination of the nodal basis functions such that, for linear Lagrange elements on triangles,

$$u^K = \sum_{i=1}^{3} N_i(u^K)\phi_i^K. \tag{37}$$

Definition 23 (Nodal Basis [12])**.** *A basis* $\{\phi_1^K, \ldots, \phi_n^K\}$ *for* $\mathcal{P}$ *is called a nodal basis if it satisfies* $N_i(\phi_j^K) = \delta_{ij}$.

Finally, to express the global function space V_h in terms of the set of finite elements, we need to introduce a local-to-global mapping. The role of the local-to-global map is to piece together the local function spaces into a global approximation in a continuous fashion. For each mesh element K, the local degree of freedom is mapped to a global degree of freedom by the mapping $\iota_K : \{1, 2, 3\} \to \{1, 2, \ldots, N\}$ where N is the total number of degrees of freedom. This means that we can define the global degrees of freedom by

$$N_{\iota_K(i)}(u) = N_i^K(u^K), \quad i = 1, \ldots, N. \tag{38}$$

We choose the local-to-global map such that if two local degrees of freedom are mapped to the same global degree of freedom then they must share the same value for each function $u \in V_h$, enforcing continuity. We can now also define the local and global interpolation operators. The global interpolation operator is used in (17) to interpolate the initial data onto the finite element space.

Definition 24 (Local Interpolation Operator [12])**.** *For a function space* V *and finite element* $(K, \mathcal{P}, \mathcal{N})$, *we define the local interpolation operator* $\mathcal{I}_k : V \to \mathcal{P}$ *such that interpolant* $\mathcal{I}_K$ *matches the original function* u *at the degrees of freedom:*

$$\mathcal{I}_K : u \mapsto \mathcal{I}_K u, \tag{39}$$

$$N_i(\mathcal{I}_K u) = N_i(u) \quad \forall N_i \in \mathcal{N}. \tag{40}$$

Definition 25 (Global Interpolation Operator [12])**.** *Let* V_h *be a function space constructed by finite elements. We define the global interpolation operator* $\mathcal{I}_h : V \to V_h$ *such that*

$$(\mathcal{I}_h u|)_K = \mathcal{I}_K(u|_K). \tag{41}$$

8.2 Assembly Process

Now we consider how to compute the mass $\mathcal{M}$ and stiffness $\mathcal{A}$ matrices. An important insight here is that the finite element basis functions have local support so these

matrices will be sparse. Hence, instead of iterating through all pairs of basis functions, it is computationally more efficient to visit each mesh element and loop over each permutation of the basis functions belonging to the mesh element to compute a local mass and stiffness matrix. The local-to-global map can then be used to add these local contributions to the global mass and stiffness matrix.

To do this, we define a reference element $(\hat{K}, \hat{\mathcal{P}}, \hat{\mathcal{N}})$ with vertices $(0,0)$, $(1,0)$, and $(0,1)$. The nodal basis of the reference element is then given by

$$\phi_1^{\hat{K}}(\hat{x},\hat{y}) = 1 - \hat{x} - \hat{y}, \quad \phi_2^{\hat{K}}(\hat{x},\hat{y}) = \hat{x}, \quad \text{and} \quad \phi_3^{\hat{K}}(\hat{x},\hat{y}) = \hat{y}, \tag{42}$$

where hatted variables represent the reference coordinates. We define a diffeomorphism F_K such that all mesh elements K can be expressed in terms of the reference element $\hat{K}$ by

$$K = F_K(\hat{K}). \tag{43}$$

Following [19], one such map is the affine transformation given by

$$\mathbf{x} = F_K(\hat{\mathbf{x}}) = \begin{pmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \mathbf{B_K}\hat{\mathbf{x}} + \mathbf{b_K}. \tag{44}$$

8.2.1 Assembly of Local Stiffness Matrix

Now we have defined a mapping from the reference element to any element in the mesh we can compute the local stiffness matrix. The entries of the local stiffness matrix are given by

$$\mathcal{A}_{ij}^K = \kappa \int_K \nabla\phi_j^K \cdot \nabla\phi_i^K \, \mathrm{d}\mathbf{x}. \tag{45}$$

We then perform a change of coordinates to the reference cell noting that $\phi^K(x,y)) = \phi^{\hat{K}}(F_K^{-1}(x,y)) =$ and therefore, by the chain rule, $\nabla\phi^K = \mathbf{B_K}^{-T}\hat{\nabla}\phi^{\hat{K}}$. Furthermore, by elementary vector calculus, we have $\mathrm{d}\mathbf{x} = |\det \mathbf{B_K}| \, \mathrm{d}\hat{\mathbf{x}}$. Hence, (45) can be written as

$$\kappa \int_K \nabla\phi_j^K \cdot \nabla\phi_i^K \, \mathrm{d}\mathbf{x} = \kappa \int_{\hat{K}} (\mathbf{B_K}^{-T}\hat{\nabla}\phi_j^{\hat{K}}) \cdot (\mathbf{B_K}^{-T}\hat{\nabla}\phi_i^{\hat{K}}) |\det \mathbf{B_K}| \, \mathrm{d}\hat{\mathbf{x}}. \tag{46}$$

We note that since we are using linear Lagrange elements, $\hat{\nabla}\phi^{\hat{K}}$ is constant and therefore can be brought outside of the integral, leaving an integrand of one. This means the integral is reduced to computing the area of the reference element, which by elementary geometry is a half. Consequently, the entries of the local stiffness matrix are therefore given by

$$\mathcal{A}_{ij}^K = \frac{\kappa}{2} |\det \mathbf{B_K}| (\mathbf{B_K}^{-T}\hat{\nabla}\phi_j^{\hat{K}})^T (\mathbf{B_K}^{-T}\hat{\nabla}\phi_i^{\hat{K}}). \tag{47}$$

Since we have already defined the nodal basis, the gradients in this expression can be computed ahead of time and are given by

$$\hat{\nabla}\phi_1^{\hat{K}} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad \hat{\nabla}\phi_2^{\hat{K}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \text{and} \quad \hat{\nabla}\phi_3^{\hat{K}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{48}$$

Then it is simply a question of tabulating the entries of the local stiffness matrix. We can exploit the symmetry of (47) while doing this.

8.2.2 Assembly of Local Mass Matrix

The entries of the local mass matrix are given by

$$\mathcal{M}_{ij}^K = \int_K \phi_j^K \phi_i^K \, \mathrm{d}\mathbf{x}, \tag{49}$$

which when mapped to the reference finite element by a similar process to that in Section 8.2.1 gives

$$\int_K \phi_j^K \phi_i^K \, \mathrm{d}\mathbf{x} = \int_{\hat{K}} \phi_j^{\hat{K}} \phi_i^{\hat{K}} |\det \mathbf{B_K}| \, \mathrm{d}\hat{\mathbf{x}}. \tag{50}$$

These are all known quantities that we can explicitly compute ahead of time, see Appendix A for details, which gives the local mass matrix as

$$\mathcal{M}^K = \frac{1}{24} |\det \mathbf{B_K}| \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}. \tag{51}$$

8.2.3 Assembly of Local Forcing Vector

The entries of the local forcing vector are given by

$$\mathcal{F}_i^K(t) = \int_K f(x, y, t) \phi_i^K \, \mathrm{d}\mathbf{x}, \tag{52}$$

which when mapped to the reference element by the same process as before gives

$$\int_K f(x, y, t) \phi_i^K \, \mathrm{d}\mathbf{x} = \int_{\hat{K}} f(T_K(\hat{x}, \hat{y}), t) \phi_i^{\hat{K}} |\det \mathbf{B_K}| \, \mathrm{d}\hat{\mathbf{x}}. \tag{53}$$

The computation of this quantity requires a quadrature method as f is a user-supplied function so we cannot compute the integral ahead of time. The most basic form of quadrature would be to take the value of f at the midpoint of the mesh element. However, to reduce the numerical error introduced, instead, we opt for a fourth-order, six-point quadrature scheme as shown in [8]. This approximation gives

$$\int_{\hat{K}} f(T_K(\hat{x}, \hat{y}), t) \phi_i^{\hat{K}} |\det \mathbf{B_K}| \, \mathrm{d}\hat{\mathbf{x}} \approx |\det \mathbf{B_K}| \sum_{i=1}^{6} w_i f(T_K(\xi_i, \eta_i), t) \phi_i^{\hat{K}}(\xi_i, \eta_i), \tag{54}$$

where w_i are the quadrature weights and (ξ_i, η_i) are the quadrature points. The quadrature weights are given by $w_1 = w_2 = w_3 = 0.111690794839005$ and $w_4 = w_5 = w_6 = 0.054975871827661$. The quadrature points by $(\xi_1, \eta_1) = (a, a)$, $(\xi_2, \eta_2) = (1 - 2a, a)$, $(\xi_3, \eta_3) = (a, 1 - 2a)$, $(\xi_4, \eta_4) = (b, b)$, $(\xi_5, \eta_5) = (1 - 2b, b)$, and $(\xi_6, \eta_6) = (b, 1 - 2b)$, where $a = 0.445948490915965$ and $b = 0.054975871827661$. We note that there are many possible choices of quadrature method that could be used here, many with better error convergence. However, given the low order of the finite elements used, this scheme was deemed appropriate. We refer the interested reader to [6] for details of other methods.

8.2.4 Assembly of Global Mass Matrix, Stiffness Matrix and Forcing Vector

Now that we can compute the local mass matrix, stiffness matrix and forcing vector, we need to combine their contributions to get the equivalent global objects. This is performed by iterating through each mesh element K in the mesh $\mathcal{T}_h$, computing the local mass and stiffness matrices and local forcing vector, and using the local-to-global map to add these local contributions to their respective global matrices. This is detailed in Algorithm 9.1.2 in [12].

9 Numerical Results

In this section, we show numerical results that verify that the scheme presented works as intended. Figure 3 shows an example solution to a homogeneous problem with Gaussian initial conditions solved using the scheme. Here we observe qualitatively the correct behaviour as the initial Gaussian profile is smoothed out. However, this gives no indication of the error in the approximation of the solution.

9.1 Manufactured Solution

To compute the error committed by the scheme, we need analytical solutions to (1) to compare the solution against. An easy way of generating analytical solutions is using the method of manufactured solutions. In this process, we guess a solution form and then choose the forcing term accordingly so that the solution satisfies the diffusion equation with homogeneous Dirichlet boundary conditions. Table 1 in Appendix C shows the computed solutions using this method for both a unit square, right angle triangle, and irregular pentagon domain. Examples of these domains are shown in

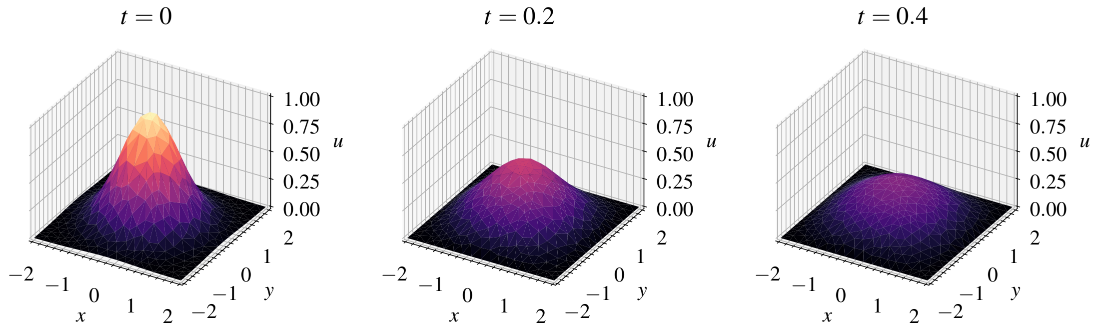

Figure 3: Evolution of the solution to the diffusion equation on the domain $\Omega = [-2,2]^2$, with homogeneous Neumann boundary conditions, $f \equiv 0$, and initial condition $u_0(x,y) = \exp\left(-(x^2+y^2)\right)$.

Figure 1, Figure 6, and Figure 7 respectively. SymPy [16] was used to symbolically compute the required forcing term.

9.2 Convergence Results

To compare the convergence of the numerical results against Theorem 18 we need to compute the error in the $\|\cdot\|_{\ell^2([0,T],H^1(\Omega))}$ norm. This involves computing the H^1 norm at each timestep and then taking the ℓ^2 norm of the resulting values. The H^1 norm is defined by

$$\|\cdot\|_{H^1(\Omega)} = \left(\int_\Omega |\cdot|^2 \,\mathrm{d}\mathbf{x} + \int_\Omega |\nabla \cdot|^2 \,\mathrm{d}\mathbf{x}\right)^{\frac{1}{2}}. \tag{55}$$

To compute the integral terms we use the same quadrature scheme defined in Section 8.2.3 as this is a higher-order quadrature scheme than the order of the finite elements used so does not pollute the convergence results.

To measure the convergence of the scheme, we decrease h, the maximum edge length in the mesh, and then compute the error in the solution when compared against the manufactured solution. We also set $\Delta t = h$ as without this it is difficult to observe the expected error convergence. This is because, if either h or Δt is kept fixed while the other is refined, the dominant term in the error expression changes, obscuring the expected convergence result. The convergence results for the scheme are shown in Figure 4. As predicted by Theorem 18, we observe linear convergence for the three different domains. This also provides a degree of confidence that the scheme is implemented correctly as we would not expect to see the correct rate of convergence if

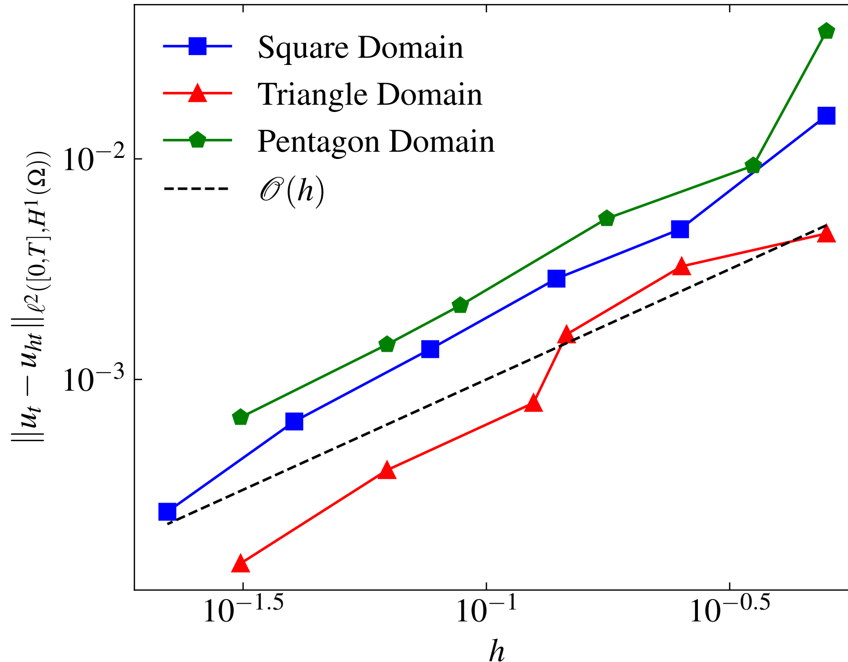

Figure 4: Error of the scheme in the $\ell^2([0,T], H^1(\Omega))$ compared against the manufactured solutions in Table 1 with $\Delta t = h$.

the scheme was implemented incorrectly. In Figure 5, we report the error with respect to the computation time. This includes the time required to assemble the mass and stiffness matrix and the forcing vector and solve the resulting linear system at each timestep but not to generate the meshes or compute the error. These timings were performed on an unburdened system equipped with an 11th Gen Intel(R) Core(TM) i7-11800H and 16 GB of memory.

10 Discussion

In this report, we have only considered the possibility of first-order Lagrange finite elements. These are the simplest finite elements to implement, however, as seen in Theorem 18, this limits the scheme to linear convergence. This would be problematic if a highly accurate solution was required as it would require the mesh resolution to be prohibitively small. For example, Figure 5 shows that the computation time for a modest error of 10^{-3} is approximately 100 seconds, which would only be worse for a larger mesh, as might be required in an engineering application. Using a quadratic Lagrange element would allow for up to second-order convergence in space and higher-

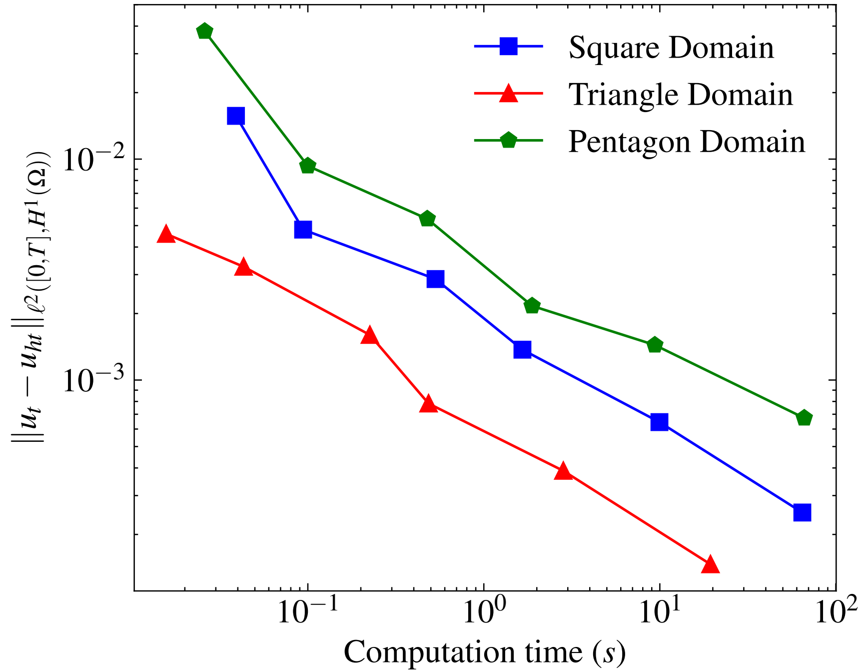

Figure 5: Computation time required to achieve a given error in the $\ell^2([0,T], H^1(\Omega))$ norm with $\Delta t = h$.

order elements would allow for even faster convergence thus permitting a coarser mesh to be used. However, when high accuracy is not required first order elements may still be a good choice as they are computationally cheaper to compute with.

In the scheme presented, the time derivative was discretised using an implicit Euler scheme, chosen due to its simplicity. It was chosen over an explicit Euler scheme as it is unconditionally stable, as shown in Theorem 17, whereas the timestep for the explicit Euler scheme must be chosen to satisfy a Courant–Friedrichs–Lewy (CFL) condition. Like the spatial discretisation, the implicit Euler scheme is first-order so a high-accuracy solution may require a prohibitively small timestep. One of the great advantages of the method of lines discretisation described in Section 6 is that the resulting system of ODEs can be solved with most numerical methods for differential equations. For example, a second-order backward difference could be used instead to achieve second-order convergence in time. To further improve the efficiency in calculating the solution an adaptive timestep could also be employed. This would allow the scheme to take small steps to preserve accuracy when the solution is changing rapidly and take large steps to speed up computation when the solution is changing slowly. Since these schemes are far more sophisticated it would make sense to use

an ‘off-the-shelf’ implementation such as those provided in PETSc [1] or SUNDIALS [14].

A further extension to the problem presented here would consider the forcing function f to be a nonlinear function of the solution u (a reaction-diffusion equation). Equations of this type are highly relevant in mathematical biology where they form the basis of the theory of pattern formation [20]. Due to the nonlinear nature of these equations, this would require modification to the scheme presented here. The first possible extension would be to use the Newton-Kantorovich method to invert the nonlinear system of equations at each timestep [12]. The second possible extension would be to treat the spatial discretisation implicitly but the nonlinear terms explicitly. This avoids the timestep restrictions that a CFL condition would introduce and also avoids the extra computation required to invert a nonlinear system of equations. This is called an implicit-explicit (IMEX) scheme.

11 Conclusion

In this report, we have shown the well-posedness of a Galerkin approximation to the heat equation. Then we presented a finite element scheme based on first-order Lagrange elements and the implicit Euler scheme for solving the problem numerically. Finally, we present numerical experiments that show agreement between the theoretical and achieved convergence rates.

References

[1] Satish Balay et al. “Efficient Management of Parallelism in Object Oriented Numerical Software Libraries”. In: *Modern Software Tools in Scientific Computing*. Ed. by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, pp. 163–202.

[2] Igor A. Baratta et al. *DOLFINx: the next generation FEniCS problem solving environment*. preprint. 2023. DOI: `10.5281/zenodo.10447666`.

[3] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer New York, 2008. ISBN: 9780387759340. DOI: `10.1007/978-0-387-75934-0`.

[4] Haim Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer New York, 2011. ISBN: 9780387709147. DOI: 10.1007/978-0-387-70914-7.

[5] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Society for Industrial and Applied Mathematics, Jan. 2002. ISBN: 9780898719208. DOI: 10.1137/1.9780898719208.

[6] Ronald Cools. "An encyclopaedia of cubature formulas". In: *Journal of Complexity* 19.3 (June 2003), pp. 445–453. ISSN: 0885-064X. DOI: 10.1016/s0885-064x(03)00011-6.

[7] Timothy A. Davis. "Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method". In: *ACM Trans. Math. Softw.* 30.2 (June 2004), pp. 196–199. ISSN: 0098-3500. DOI: 10.1145/992200.992206.

[8] Gouri Dhatt, Gilbert Touzot, and Emmanuel Lefrançois. *Finite Element Method*. Wiley, Oct. 2012. ISBN: 9781118569764. DOI: 10.1002/9781118569764.

[9] Alexandre Ern and Jean-Luc Guermond. *Finite Elements III: First-Order and Time-Dependent PDEs*. Springer International Publishing, 2021. ISBN: 9783030573485. DOI: 10.1007/978-3-030-57348-5.

[10] Alexandre Ern and Jean-Luc Guermond. *Theory and Practice of Finite Elements*. Springer New York, 2004. ISBN: 9781475743555. DOI: 10.1007/978-1-4757-4355-5.

[11] Lawrence Evans. *Partial Differential Equations*. American Mathematical Society, Mar. 2010. ISBN: 9781470411442. DOI: 10.1090/gsm/019.

[12] Patrick E. Farrell. *C6.4 Finite Element Methods for PDEs Course Notes*. Mathematical Institute, University of Oxford, Hilary Term 2022.

[13] David A. Ham et al. *Firedrake User Manual*. First edition. Imperial College London et al. May 2023. DOI: 10.25561/104839.

[14] Alan C. Hindmarsh et al. "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software* 31.3 (Sept. 2005), pp. 363–396. ISSN: 1557-7295. DOI: 10.1145/1089014.1089020.

[15] Andreas Kloeckner et al. *MeshPy*. Version 2022.1.1. Nov. 2022. DOI: 10.5281/zenodo.7296572.

[16] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103.

[17] Jonathan Richard Shewchuk. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1996, pp. 203–222. ISBN: 9783540706809. DOI: 10.1007/bfb0014497.

[18] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer New York, 2002. ISBN: 9780387217383. DOI: 10.1007/978-0-387-21738-3.

[19] Endre Süli. *Lecture Notes on Finite Element Methods for Partial Differential Equations*. Mathematical Institute, University of Oxford, Aug. 2020.

[20] A.M. Turing. "The Chemical Basis of Morphogenesis". In: *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences* 237.641 (Aug. 1952), pp. 37–72. DOI: 10.1098/rstb.1952.0012.

[21] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (Feb. 2020), pp. 261–272. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2.

Appendices

A Computation of the Entries of the Local Mass Matrix

Here we compute the entries of the local stiffness matrix (whilst exploiting its symmetry):

$$\mathcal{M}_{11}^K = |\det \mathbf{B_K}| \int_{\hat{K}} (1 - \hat{x} - \hat{y})^2 \,\mathrm{d}\hat{\mathbf{x}} = \frac{1}{12}|\det \mathbf{B_K}|, \tag{56}$$

$$\mathcal{M}_{12}^K = \mathcal{M}_{21}^K = |\det \mathbf{B_K}| \int_{\hat{K}} (1 - \hat{x} - \hat{y})\hat{x} \,\mathrm{d}\hat{\mathbf{x}} = \frac{1}{24}|\det \mathbf{B_K}|, \tag{57}$$

$$\mathcal{M}_{13}^K = \mathcal{M}_{31}^K = |\det \mathbf{B_K}| \int_{\hat{K}} (1 - \hat{x} - \hat{y})\hat{y} \,\mathrm{d}\hat{\mathbf{x}} = \frac{1}{24}|\det \mathbf{B_K}|, \tag{58}$$

$$\mathcal{M}_{22}^K = |\det \mathbf{B_K}| \int_{\hat{K}} \hat{x}^2 \,\mathrm{d}\hat{\mathbf{x}} = \frac{1}{12}|\det \mathbf{B_K}|, \tag{59}$$

$$\mathcal{M}_{23}^K = \mathcal{M}_{32}^K = |\det \mathbf{B_K}| \int_{\hat{K}} \hat{x}\hat{y} \,\mathrm{d}\hat{\mathbf{x}} = \frac{1}{24}|\det \mathbf{B_K}|, \tag{60}$$

$$\mathcal{M}_{33}^K = |\det \mathbf{B_K}| \int_{\hat{K}} \hat{y}^2 \,\mathrm{d}\hat{\mathbf{x}} = \frac{1}{12}|\det \mathbf{B_K}|. \tag{61}$$

$$\tag{62}$$

B Additional Example Meshes

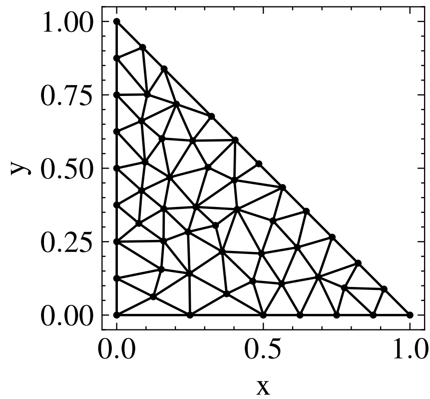

Figure 6: Example mesh for the right angle triangle domain.

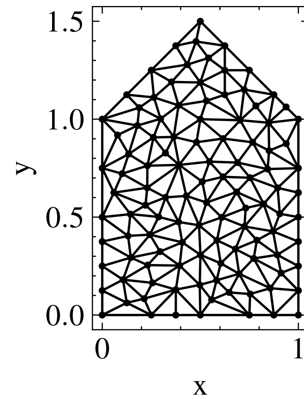

Figure 7: Example mesh for the irregular pentagon domain.

C Manufactured Solutions

Domain	Analytical Solution $u(x,y,t)$	Forcing Term $f(x,y,t)$
Unit square	$xy\,(x-1)\,(y-1)\,e^t$	$(-2\kappa(x(x-1)+y(y-1))$ $+xy(x-1)(y-1))e^t$
Right angle triangle	$xy\,(-x-y+1)\,e^t$	$(-xy\,(x+y-1)+2x+2y)\,e^t$
Irregular Pentagon Domain	$xy(x-1)(-x+y-1)(x+y-2)e^t$	$(-2\kappa(3x(x-1)(y-1)$ $-y(x(x-1)+x(x-y+1)$ $+x(x+y-2)$ $+(x-1)(x-y+1)$ $+(x-1)(x+y-2)$ $+(x-y+1)(x+y-2)))$ $-xy(x-1)(x-y+1)(x+y-2))e^t$

Table 1: Analytical solution for different domains generated using the method of manufactured solutions.

D Python Implementation

```
from typing import Optional

import numpy as np
import scipy
from meshpy.tet import MeshInfo
from tqdm import tqdm


def calc_transform_mat(vertices: np.ndarray) -> np.ndarray:
    """Calculate the affine transformation matrix from the reference
   element to
    the element.

    Parameters
    ----------
    vertices : np.ndarray
        The vertices of the element

```

```python
    Returns
    -------
    np.ndarray
        Affine transformation matrix
    """
    B_k = np.array(
        [
            vertices[1, :] - vertices[0, :],
            vertices[2, :] - vertices[0, :],
        ]
    ).T
    return B_k


def local_stiffness_matrix(vertices: np.ndarray) -> np.ndarray:
    """Assemble the local stiffness matrix of a 2d CG1 element.

    Parameters
    ----------
    vertices : np.ndarray
        Coordinates of the nodes of the element.

    Returns
    -------
    np.ndarray
        local stiffness matrix of the element.
    """
    B_k = calc_transform_mat(vertices)
    grads = np.array([[-1, 1, 0], [-1, 0, 1]])
    area = 0.5 * np.linalg.det(B_k)
    S = np.empty((3, 3))
    for i in range(3):
        for j in range(3):
            S[i, j] = np.linalg.solve(
                B_k.T, grads[:, j]
            ).T @ np.linalg.solve(B_k.T, grads[:, i])
    return area * S


def local_mass_matrix(vertices: np.ndarray) -> np.ndarray:
    """Assemble the local mass matrix of a 2d CG1 element.

    Parameters
```

```
61      ----------
62      vertices : np.ndarray
63          Coordinates of the nodes of the element.
64
65      Returns
66      -------
67      np.ndarray
68          local mass matrix of the element.
69      """
70      tmp = scipy.linalg.toeplitz([2, 1, 1])
71      B_k = calc_transform_mat(vertices)
72      return np.linalg.det(B_k) * tmp / 24
73
74
75  def local_forcing_vector(
76      f: callable, vertices: np.ndarray, t: float
77  ) -> np.ndarray:
78      """Compute the local forcing vector for an element using three
79      point quadrature.
80
81      Parameters
82      ----------
83      f : callable
84          forcing function
85      vertices : np.ndarray
86          vertices of the element
87      t : float
88          current time
89
90      Returns
91      -------
92      np.ndarray
93          local forcing vector
94      """
95      basis_func = (
96          lambda x, y: 1 - x - y,
97          lambda x, y: x,
98          lambda x, y: y,
99      )
100     # constants for defining quad points
101     a = 0.445948490915965
102     b = 0.091576213509771
103     # quadrature points
```

```
104     quad_points = np.array(
105         [
106             [a, a],
107             [1 - 2 * a, a],
108             [a, 1 - 2 * a],
109             [b, b],
110             [1 - 2 * b, b],
111             [b, 1 - 2 * b],
112         ]
113     )
114     # quadrature weights
115     weights = np.array(
116         [
117             0.111690794839005,
118             0.111690794839005,
119             0.111690794839005,
120             0.054975871827661,
121             0.054975871827661,
122             0.054975871827661,
123         ]
124     )
125     B_k = calc_transform_mat(vertices)
126     c = vertices[0, :]
127     b = np.empty(3)
128     for j in range(3):
129         tmp = 0
130         for i, w in enumerate(weights):
131             # map reference coords to really coordinates
132             eval_point = B_k @ quad_points[i, :] + c
133             tmp += f(*eval_point, t) * basis_func[j](*eval_point) * w
134         b[j] = tmp
135     return np.linalg.det(calc_transform_mat(vertices)) * b
136 
137 
138 def solve_heat_eq(
139     mesh: MeshInfo,
140     f: callable,
141     T: float,
142     dt: float,
143     u0: callable,
144     kappa: Optional[float] = 1,
145 ):
146     """Solves the equation heat equation using CG1 finite elements.
```

```
147
148     More specifically solves:
149         u_t = div grad u + f(x,t) on omega*(0,T)
150         u = 0                     on gamma
151
152     Parameters
153     ----------
154     mesh : MeshInfo
155         `meshpy` mesh object to solve the problem on.
156     f : callable
157         Forcing function. Must have the signature ``f(x, y, t)``.
158     T : float
159         Final time to solve to.
160     dt : float
161         Time step.
162     u0 : callable
163         Initial conditions. Must have signature ``u0(x, y)``.
164     kappa : float, optional
165         Coefficient of diffusivity, by default 1
166     """
167     # initialisation
168     # coordinates of all the nodes
169     nodes = np.asarray(mesh.points)
170     # number of time steps
171     N = int(T // dt)
172     # number of nodes
173     n_nodes = nodes.shape[0]
174     # node id of all nodes on the boundary
175     # as we have Dirichlet BCs we do not need to solve for these
176     boundary_nodes = np.unique(mesh.facets)
177     # mask for extracting the free nodes
178     free_nodes = np.delete(np.arange(n_nodes), boundary_nodes)
179     # node ids that comprise each element
180     elements = np.asarray(mesh.elements)
181     # solution array
182     U = np.zeros((N + 1, n_nodes))
183     # stiffness matrix
184     A = scipy.sparse.dok_array((n_nodes, n_nodes))
185     # mass matrix
186     B = scipy.sparse.dok_array((n_nodes, n_nodes))
187
188     # initial conditions
189     U[0, :] = u0(nodes[:, 0], nodes[:, 1])
```

```
190
191     # assemble stiffness matrix and mass matrix
192     for element in elements:
193         A[element[:, np.newaxis], element] += local_stiffness_matrix(
194             nodes[element]
195         )
196         B[element[:, np.newaxis], element] += local_mass_matrix(
197             nodes[element]
198         )
199
200     # convert to CSC format for better performance
201     A = A.tocsc()
202     B = B.tocsc()
203
204     for n in tqdm(range(1, N + 1)):
205         b = np.zeros(n_nodes)
206
207         # rhs forcing
208         for element in elements:
209             b[element] += dt * local_forcing_vector(
210                 f, nodes[element], n * dt
211             )
212
213         # previous solution
214         b += B @ U[n - 1, :]
215
216         lhs = (
217             B[free_nodes[:, np.newaxis], free_nodes]
218             + dt * kappa * A[free_nodes[:, np.newaxis], free_nodes]
219         )
220
221         U[n, free_nodes] = scipy.sparse.linalg.spsolve(
222             lhs, b[free_nodes], use_umfpack=True
223         )
224
225     return U, nodes, dt * np.arange(N + 1), A, B
```