Imperial College London

MEng Mathematics & Computer Science
Final Year Project Report

Department of Computing

Imperial College of Science, Technology and Medicine

Variational Time-Steppers that are Finite Element in Time for Firedrake and Irksome

Supervisor:
Colin Cotter

Author:
Giacomo La Scala

Second Marker:
David Ham

June 22, 2022

Submitted in partial fulfillment of the requirements for the MEng Mathematics and Computer Science degree of Imperial College London

Abstract

Partial differential equations (PDEs) model an immense variety of physical phenomena of theoretical and practical interest, although they often do not present closed-form solutions. Firedrake is a high-performance, automated code generation software tool to approximate the solution to PDEs using the finite element method (FEM). Time-dependent PDEs are commonly solved numerically by applying the FEM only in the spatial-domain, reducing the problem to a number of time-dependent ordinary differential equations solved by time-stepping methods like Runge-Kutta. This report presents Fetsome, an extension to Firedrake and the Irksome time-stepping library which discretises the time domain through finite elements like in the spatial domain, providing a new software implementation for the mathematical formulation of finite element in time (FET). Fetsome supports continuous Petrov-Galerkin and discontinuous Galerkin time elements, which decouple the problem defined on the global time-mesh to obtain local FET time-stepping. Code verification is performed through convergence testing for classical linear and nonlinear time dependent PDEs such as the heat, transport and Burger's equations. By analysing the stability properties of solutions to the inviscid Burgers' equation, the efficacy of the implementation in combining with more complex spatial discretisations and flux functions is highlighted.

Acknowledgements

First of all, I would like to thank my supervisors Colin Cotter and David Ham for giving me the opportunity of taking this project despite not originally proposing it. Their help for every part of this project has been invaluable and their constant availability, even across many timezones, has been something I am very grateful for.

I would also like to thank the Firedrake team, in particular Daniel Shapero, Robert Kirby and Lawrence Mitchell, for their kind assistance with a variety of Firedrake and Irksome related matters. Being able to ask a quick question at any time has been immensely helpful.

Last but definitely not least, I thank the whole of my family for their incessant support, at any hour of the day or of the night, from all corners of Italy and beyond. Grazie veramente. And thank you too.

Contents

Chapter 1

Introduction

Partial differential equations (PDEs) are a fundamental tool in modelling the behaviour of evolving systems and form one of the most important research areas in mathematics, both from a theoretical and an applied point of view. The applications of PDEs are countless: from the Korteweg-de Vries equation for water wave solitons, through the Klein-Gordon equation in relativistic fundamental particles, or Skellam's reaction-diffusion model for population dispersal [1], to name a few. It is therefore easy to understand the profound impact of PDEs on scientific and engineering fields of any kind.

The relevance and power of PDEs is often hindered either by the impossibility of obtaining closed form solutions to such problems or by the complex domains on which the problems are posed. The Finite Element Method (FEM) is one of the most common and powerful numerical methods that are employed to overcome this problem and obtain solutions to PDEs on arbitrarily complex domains. It attempts to approximate these domains through "meshes" of connected nodes, partitioned into elements, over which a solution basis is defined. The PDE of interest is then manipulated into its weak variational form and its approximate solution is sought in this basis. Many software packages present an implementation of the tools needed to solve finite element (FE) problems. Such packages aspire to provide great accuracy with great performance, usually measured by their temporal and energetic efficiencies in obtaining numerical solutions, as well as a fundamental simplicity-of-use for users who are often not acquainted with writing high-performance, at times even parallel, mathematical code.

Firedrake [2] is one of such software packages and the one on which this study focuses. Firedrake provides a framework for automated generation of low-level, high-performance finite element code that can be used by scientists, engineers and mathematicians alike, through its focus on providing a natural software extension to the mathematical theory of the FEM.

The FEM approach of discretising spatial domains through a mesh of simplexes (or polytopes) is widely used. Nevertheless, many problems of interest that also involve time derivatives are not usually approached by preparing a finite element in time (FET) discretisation, which uses higher dimensional space-time meshes. In fact, time-dependent PDEs are usually only discretised in space, to obtain ordinary differential equations that can be solved iteratively by conventional time stepping methods. One of the most widely adopted of these methods, also considered in this study, is that of generic Runge-Kutta (RK) time stepping.

Many important FE software frameworks, such as FEniCS [3] and DUNE [4], reflect this: a user can employ the provided functionality to manipulate their problem, spatial domain and boundary conditions but their code must be integrated with manually-implemented loops that evolve the solutions forward through time. Besides introducing the need for implementation specific knowledge on the side of the users, this forfeits the possibility of maintaining a higher level of mathematical abstraction. This would often be useful in allowing a more natural and generic software composition of the elements that constitute a numerical method for solving PDEs. In the case of traditional RK time-steppers, the Irksome [5] extension of Firedrake begins to challenge this. It provides its users with a symbolic definition of the time derivative, subsequently interpreted inside expressions by a symbolic manipulation layer that composes and solves the associated Runge-Kutta problems separately.

Nevertheless, Irksome does not support FET. Besides yielding a more unified formulation for the solution of time-dependent PDEs, a FET solver is desirable in its automatic generation of high-order time-stepping schemes. Also, solutions produced with FET are always meaningful over the continuous time domain. Furthermore, FET presents the opportunity to apply spatial FEM techniques, such as choice of quadrature, in the time domain to investigate their effects on efficiency and accuracy for time-dependent problems.

The implementation presented in this report extends Firedrake's and Irksome's symbolic manipulation layer with Fetsome, a novel FET time-stepping layer that solves PDEs using space-time finite elements. The focus of this implementation is on enabling FET solutions to fundamental time-dependent problems, such as the heat equation and Burgers' equation, commonly found in all fields of applied mathematics. The goals of Fetsome and this investigation therefore are to:

- Introduce an automated FET layer for solving space-time dependent linear and nonlinear PDEs, integrated with Firedrake and Irksome
- Support continuous (Petrov-Galerkin) and discontinuous (Galerkin) discretisations of the time domain
- Support time-dependent forcing functions and spatial boundary conditions in FET
- Demonstrate FET in the context of solving classical time-dependent PDEs
- Evaluate the convergence properties of the supported time finite elements for code verification

This report describes all layers involved in the development of Fetsome, from the derivation of the mathematical formulations and algebraic systems, through the implementation, to showcasing and analysing the developed functionality.

Chapter 2

Mathematical and Computational Background

2.1 Finite Element in Space

This section introduces the finite element method by considering the Poisson problem posed on an exclusively spatial domain. It is shown how, by formulating the weak variational problem on an approximating space, Firedrake can be used to obtain a numerical solution converging to its exact solution.

2.1.1 A Standard Example

Let $\Omega \subset \mathbb{R}^2$ be a two-dimensional spatial domain, then the forced Poisson problem requires finding $u(x, y)$ such that the following equation, combined to specific boundary conditions, is satisfied:

$$-\nabla^2 u = f(x, y) \tag{2.1.1}$$

In particular, suppose that $\Omega = [0, 1] \times [0, 1]$ (the unit square) and that the forcing function is $f(x, y) = 5\pi^2 \sin(\pi x) \sin(2\pi y)$. Suppose also that vanishing Dirichlet boundary conditions are enforced such that $u|_{\partial\Omega} = 0$. It can be verified that $u(x, y) = \sin(\pi x) \sin(2\pi y)$ solves the equation. Now let $\mathcal{T}$ be a triangulation (forming a "mesh") of the domain Ω and let V_h be a function space such that the restriction of $v \in V_h$ to a triangle $K \in \mathcal{T}$ is a degree k polynomial. To obtain the variational formulation of the Poisson equation for what is called the trial function $u \in V_h$, equation (2.1.1) must be multiplied by an arbitrary test function $v \in V_h$ and integrated by parts over Ω:

$$\int_\Omega -v\nabla^2 u dx = \int_\Omega f v dx \tag{2.1.2}$$

$$\implies \sum_{K\in\mathcal{T}} \left[\int_K \nabla v \cdot \nabla u dx - \int_{\partial K} v \nabla u \cdot \mathbf{n} dS \right] = \sum_{K\in\mathcal{T}} \int_K f v dx \tag{2.1.3}$$

$$\implies \int_\Omega \nabla v \cdot \nabla u dx - \int_{\partial\Omega} v \nabla u \cdot \mathbf{n} dS - \int_\Gamma v \nabla u \cdot (\mathbf{n}^+ + \mathbf{n}^-) dS = \int_\Omega f v dx \tag{2.1.4}$$

Where Γ is the set of all triangle edges that are contained in the interior of Ω and not on its boundary. As all interior edges are shared by two triangles, each of these will contribute to the surface integral through the $\mathbf{n}^+$ and $\mathbf{n}^-$ terms. As the function space is continuous

across triangles, the boundary term can be factored from its dot product with the normals in this same integral. Note that the two normals are equal and opposite by definition, as they belong to the same edge from opposite facing triangles, so the whole interior edge integral vanishes. To enforce the vanishing Dirichlet boundary condition, the space of solutions and test functions V_h is modified to $\tilde{V}_h = \{v \in V_h | v|_{\partial\Omega} = 0\}$ such that we now take $u, v \in \tilde{V}_h$. Therefore, by $v|_{\partial\Omega} = 0$, the integral over $\partial\Omega$ also vanishes, leaving:

$$\forall v \in \tilde{V}_h : \quad \int_\Omega \nabla v \cdot \nabla u dx = \int_\Omega f v dx \tag{2.1.5}$$

From equation (2.1.5) two forms can be defined, one bilinear in u and v and one linear in v. Through this, the weak variational problem for the Poisson equation is to find $u \in \tilde{V}_h$ such that for all $v \in \tilde{V}_h$:

$$b(v, u) = L(v) \tag{2.1.6}$$

$$b(v, u) = \int_\Omega \nabla v \cdot \nabla u dx \tag{2.1.7}$$

$$L(v) = \int_\Omega 5\pi^2 \sin(\pi x) \sin(2\pi y) v dx \tag{2.1.8}$$

If a basis $\{\phi_i(x, y)\}_i$ of $\tilde{V}_h$ is taken, the following is obtained by imposing equation (2.1.6) is satisfied for all basis functions (by linearity in v):

$$\forall \phi_i(x, y) : \quad \sum_j u_j b(\phi_i, \phi_j) = L(\phi_i) \tag{2.1.9}$$

$$\implies B\mathbf{u} = \mathbf{L} \tag{2.1.10}$$

Where $B_{ij} = b(\phi_i, \phi_j)$, $\mathbf{u}_j = u_j$ are the constant coefficients of u expanded in the basis, $\mathbf{L}_i = L(\phi_i)$. Therefore the finite element approximation can be solved by reducing to such a matrix-vector system. It is worth noting that, in general, bilinear forms correspond to matrices and linear forms to vectors when deriving the algebraic system for a problem. With the weak variational formulation for the Poisson problem on $\tilde{V}_h$ complete, it is now possible to seek a solution using Firedrake.

2.1.2 Firedrake

Firedrake [2] is the automated system for solving PDEs with the finite element method on which this study is based. It is a freely-available Python package that aims to present a unified framework spanning from the definition of weak variational forms and problems on arbitrary meshes, to the assembly of their equivalent systems of equations (like (2.1.9)), to their high-performance sequential and parallel solutions.

On the highest level, Firedrake uses an extended version of the UFL [6] (Unified Form Language, see 2.4.1) domain specific language (maintained by the FEniCS [3] Project) to describe the integrals (such as (2.1.6)) over spatial domains that are employed in the FEM. Finite element problems are posed on meshes and Firedrake supports both building meshes programmatically as well as loading unstructured meshes from tools such as Gmsh [7]. Similarly, Firedrake presents users with simple and effective ways of defining function spaces on

arbitrary one, two or three dimensional meshes of polytopes supported by a variety of finite elements. These include, but are not limited to: continuous and discontinuous Lagrange for scalar elements, Brezzi-Douglas-Marini, Raviart-Thomas, H(div) and H(curl) for vector valued elements and the Argyris element for derivative continuity as well as more exotic choices such as the Hellan-Herrmann-Johnson and Bell elements. Functions defined on Firedrake function spaces seamlessly interact with UFL to the point that some of the changes made by Firedrake to UFL have been incorporated into its upstream release. Arbitrary functions in the coordinates of the defined meshes can also be used through their interpolation onto the supported function spaces.

In its lower levels, Firedrake reduces a finite element problem to objects and data structures that better represent the computational task by interfacing with representation, tabulation and solver libraries such as `petsc4py` [8] and FIAT/FInAT [9, 10] (see 2.4.3, 2.4.4 for more details). Subsequently, its Two-Stage Form Compiler [11] compiles the tensor algebraic representations of the specific finite element problem to efficient and explicitly-parallel C code that is architecture specific and can be executed in scalable, high-performance contexts.

Firedrake's philosophy of being flexible and composable has allowed it to be central in the development of application-specific PDE frameworks like Thetis [12], for coastal ocean modelling, and Icepack [13], for glacier flow, as well as supporting extensions like Slate [14], to represent linear algebra operations on tensors. Together with the newly developed Irksome package (see 2.4.2), these make development of a FET package for Firedrake very appealing for the vast potential of applications it can target.

2.1.3 Solution with Firedrake

We now seek to solve the Poisson problem (2.1.1) on the unit square numerically to introduce Firedrake's core functionality. Firstly, the domain of the problem is defined by providing a mesh and a corresponding function space over it. The finite element space composed of the function space over the mesh will provide both the trial and the test functions for the weak variational problem. In this case, the mesh is picked to partition a unit square and is triangulated using a 10x10 grid, with each triangle supporting a continuous Lagrange element composed of polynomials of degree less than or equal to 3 (the cubics). A traditional Lagrange element involves functions that are specified by point-evaluation at equispaced points in each triangle. In the case of Firedrake, the default `spectral` variant is a slight modification of this, picking the points at which to evaluate functions to be Gauss-Lobatto-Legendre points for better conditioning on the systems to be solved. The following then defines the mesh, function space and functions that are needed to specify the weak variational problem:

```
1 from firedrake import *
2
3 mesh = UnitSquareMesh(10, 10)
4 v = FunctionSpace(mesh, "CG", 3)
5 u = TrialFunction(V)
6 v = TestFunction(V)
```

The bilinear and linear forms that were derived in (2.1.6) can be easily translated into Firedrake-compatible UFL using syntax that closely resembles mathematical operations. The

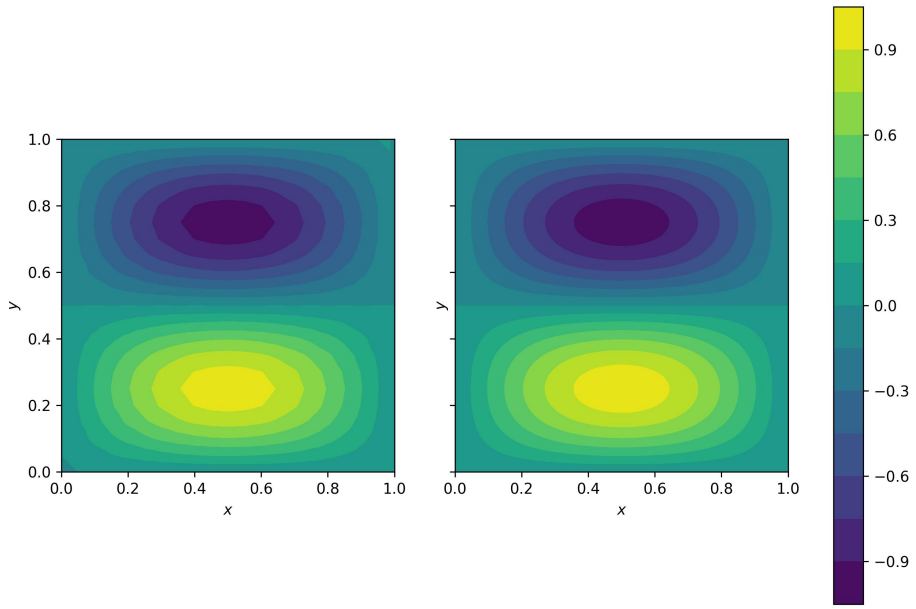


Figure 2.1: Contour plots of the finite element solution (left) on a 10×10 grid of cubic Lagrange elements on triangles, and the exact solution $\sin(\pi x)\sin(2\pi y)$ (right)

forcing function $f(x, y) = 5\pi^2 \sin(\pi x)\sin(2\pi y)$ is interpolated onto the finite element space `V` before it is used in the linear forcing form through Firedrake's `interpolate`.

```
x, y = SpatialCoordinate(mesh)
f = interpolate(5*pi**2 * sin(pi*x) * sin(2*pi*y), V)

b = dot(grad(u), grad(v)) * dx
L = f*v*dx
```

Lastly, the vanishing Dirichlet boundary condition on the unit square boundary is enforced using `DirichletBC`, specifying the domain as `"on_boundary"` to be applied on the whole domain boundary:

```
bc = DirichletBC(V, Constant(0.), "on_boundary")
```

Once the weak variational problem and its boundary condition are fully specified, we can define a solution variable and let Firedrake's `solve` compile the problem down to an algebraic system solved by PETSc:

```
uh = Function(V)
solve(b == L, uh, bcs=[bc])
```

Figure 2.1 shows the analytical solution $u(x, y) = \sin(\pi x)\sin(2\pi y)$ held in `uh` to the Poisson problem compared to the obtained numerical solution using Firedrake. If h is the maximum between the diameters of the triangles partitioning Ω, the choice of a cubic polynomial continuous Lagrange finite element space guarantees convergence of the Poisson problem in the order $O(h^4)$. Hence, this numerical solution will converge to the exact solution as the number of elements is increased uniformly, as the mesh size parameter decreases.

2.2 Finite Element in Time

Finite element in time has a well-established mathematical theory. Nevertheless, its formulation into software that can be used by general-problem finite element packages is not available in the literature. This section outlines the mathematical extension of the finite element method to space-time elements and their convergence, forming the theoretical foundations for the implementation of Fetsome, as well as a new, practical derivation of the systems that underpin its translation into automatic software.

2.2.1 Numerical Quadrature in the Time Domain

Before the standard finite element method can be extended from an exclusively spatial domain to the space-time domain, quadrature in the time domain must be presented. Quadrature is the numerical method used to approximate an integral of an integrand over a chosen domain. Quadrature approximates integrals by their finite Riemann sums up to some order. In conventional calculus, the limit of these Riemann sums as the typical size of the partition over which they are defined tends to zero would be one formal mathematical definition of an integral. The approximation of an integral over a time domain $[0, T]$ of a space-time function f through quadrature can be generically expressed as:

$$\int_0^T f(\mathbf{x}, t)dt \approx \sum_{i=1}^{q} b_i f(\mathbf{x}, t_i) \tag{2.2.1}$$

In this form, the q point scheme is composed of the t_i quadrature points and the b_i weights.

Quadrature rules can be multi-dimensional, although a fundamental property of the time domain is that it is always one-dimensional. This means that all quadrature schemes dealt with inside this study will be one-dimensional, presenting a great degree of simplification of the mathematics compared to that involved in the study of higher dimensional spatial quadratures. Furthermore, this report will refer to the degree of precision of a quadrature scheme as the highest polynomial degree that can be integrated exactly by the scheme.

When implementing a numerical solver that uses quadrature to compute time integrals, it is important to choose schemes that have both the correct degrees of precision and enforce evaluation of the integrand at the desired time points. In the case of continuous finite elements in time implemented through a Petrov-Galerkin method, introduced in section 2.2.3, a Gauss-Legendre quadrature scheme with evaluation points on the interior of the element is suggested [15]. A q point Gauss-Legendre quadrature has degree of precision of $2q - 1$. Differently, a discontinuous Galerkin method, introduced in section 2.2.4, will use right-sided Gauss-Radau quadrature schemes, which evaluate a function also at the rightmost end of a time-step and have a degree of precision $2q - 2$ for a q point quadrature [15].

2.2.2 Heat Equation Example

Using the following example, it is demonstrated how it is possible to augment the usual finite element method in the three spatial dimensions with a time discretisation. It is then shown how a block system for the problem can be obtained and solved for the solution in the chosen space-time finite element space.

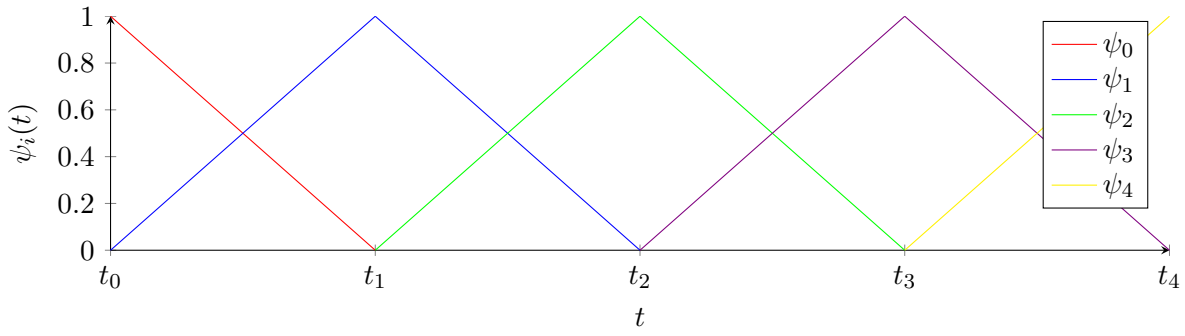


Figure 2.2: First five temporal basis functions ψ_i for the interval $[0, T]$

Consider the heat equation problem on the domain $\Omega \times [0, T]$, with Ω representing the spatial domain and $[0, T]$ the finite closed time interval over which the solution is sought:

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0 \tag{2.2.2}$$

The heat equation is equipped with the Neumann boundary condition $\frac{\partial u}{\partial n} = \nabla u \cdot \hat{\mathbf{n}} = 0$ on $\partial\Omega$, representing no heat dissipation at the boundary. From the time dependence, the problem requires an initial condition for $u(\mathbf{x}, t)$, namely $u(\mathbf{x}, 0) = \rho(x)$. It is to be noted that the thermal diffusivity is taken to be 1 and that the equation can also be referred to as a diffusion equation in the literature.

Let $\mathcal{T}$ be a polygonal partition of the spatial domain Ω such that for each $K_i \in \mathcal{T}$, $(K_i, \mathcal{P}_i, \mathcal{N}_i)$ is a Ciarlet finite element [16] with $\mathcal{P}_i$ being the space of shape functions on K_i and $\mathcal{N}_i$ being the set of nodal variables forming a basis for the dual space $\mathcal{P}_i'$. It is assumed that the partition covering Ω will not change through time. We wish to approximate the spatial solutions $u(\mathbf{x}, t)$ on Ω for a fixed $t \in [0, T]$ by any chosen finite element space $S_h \subset S$ with S a Sobolev space, h representing a maximum measure for spatial elements in the mesh. Let $\Phi = \{\phi_k(\mathbf{x})\}_{k=0}^{d}$ be a finite $d + 1$ dimensional basis for S_h.

Then, let P^N be the chosen discretisation of the one-dimensional time domain $[0, T]$ such that $P^N = \{0 = t_0 < t_1 < \ldots < t_N = T\}$ and $t_{n+1} = t_n + \Delta t$, with $\Delta t = \frac{T}{N}$. For this example, the chosen discretisation is uniformly spaced for all time-steps, but it can be easily modified to allow generically spaced points on the one-dimensional time domain. Let D be a Sobolev space and let $D_k \subset D$ ($k = \Delta t$ maximum time "mesh" size) be the finite element space that represents the temporal dependency of the approximated solution, with each interval $[t_n, t_{n+1}]$ being a time finite element. Let the temporal basis $\Psi = \{\psi_i(t)\}_{i=0}^{p}$ for D_k be formed of piecewise-continuous polynomial functions such that $\psi_i(t_j) = \delta_{ij}$. For piecewise-linear basis functions, this temporal basis is visually composed of hat functions on the interior points of P^N, formed by a linearly increasing function between t_{n-1} and t_n combined with a linearly decreasing function between t_n and t_{n+1} with range 0 to 1. Figure 2.2 presents the basis Ψ in the case of linear shape functions as a graph.

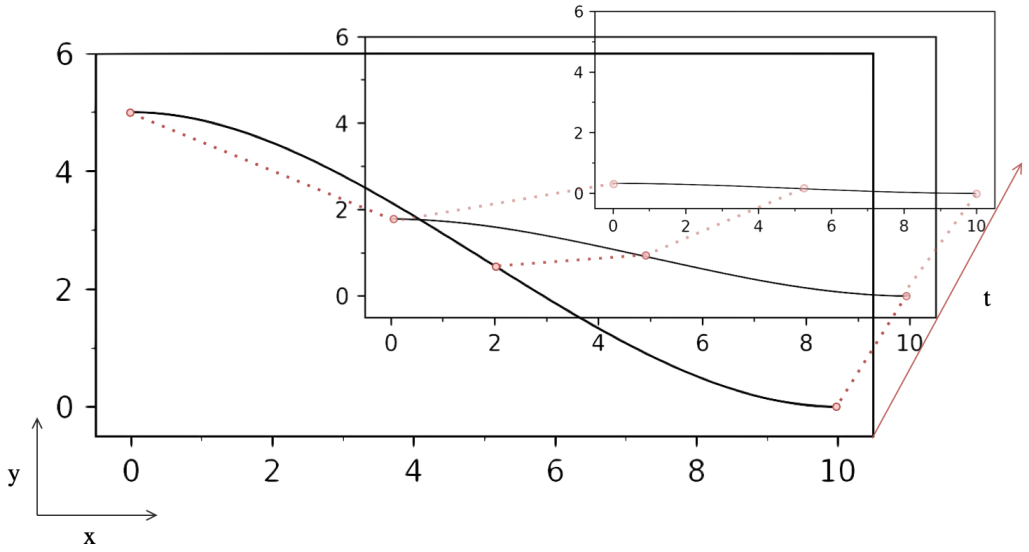


Figure 2.3: A spatial function evolving across timesteps/time-elements where each basis function is linear, showing the tensor product nature of space-time elements

Having defined spatial and temporal spaces and bases, we can define the Sobolev and Galerkin approximating spaces for the problem on $\Omega \times [0, T]$ as the tensor product spaces $W = D \otimes S$ and $W_{kh} = D_k \otimes S_h$, as justified by Aziz and Monk in [17]. From the definition, we obtain the basis functions for W_{kh}:

$$\phi_{ij}(\mathbf{x}, t) = \psi_i(t)\phi_j(\mathbf{x}) \tag{2.2.3}$$

Above, the index i is interpreted as the time-point and j is interpreted as the unique spatial node identifier. It follows that any function in the space can be expressed as a linear combination of basis functions $f(\mathbf{x}, t) = f_{ij}\phi_{ij}(\mathbf{x}, t), with f_{ij} \in \mathbb{R}$ constants, where the Einstein summation convention over indices is used for conciseness. Figure 2.3 illustrates the tensor product nature underpinning a time evolving spatial system, with the spatial dimension being replicated for all points on the time axis.

We proceed to formulate the weak variational problem by taking a test function $v \in W$ (as shown later, v need not to be from the same space as u), multiplying equation (2.2.2) and integrating on the space-time domain:

$$\begin{aligned}
&\frac{\partial u}{\partial t}v - (\nabla^2 u)v = 0 \\
\implies &\int_0^T \int_\Omega \left(\frac{\partial u}{\partial t}v - (\nabla^2 u)v\right) dxdt = \int_0^T \int_\Omega \left(v\frac{\partial u}{\partial t} + \nabla u \cdot \nabla v\right) dxdt - \int_0^T \int_{\partial\Omega} v(\nabla u \cdot \hat{\mathbf{n}}) dsdt = 0 \\
\implies &\int_0^T \int_\Omega \left(v\frac{\partial u}{\partial t} + \nabla u \cdot \nabla v\right) dxdt = 0
\end{aligned} \tag{2.2.4}$$

With the last part following from the natural Neumann boundary condition $(\nabla u \cdot \hat{\mathbf{n}})|_{\partial\Omega} = 0$. Integration by parts can be carried out by the piecewise continuity of the space-time basis functions across elements. It is important to see that this multiplication and integration guarantees that the resulting space-time equation is always linear in v. Since the heat equation

is to be satisfied for each time sub-interval, it must be that in a partition P^N for each single time-step $t_n < t < t_n + \Delta t$ we can reduce the time integral to just over this interval:

$$\int_{t_n}^{t_n+\Delta t} \int_\Omega \left(v \frac{\partial u}{\partial t} + \nabla u \cdot \nabla v \right) dx dt = 0 \tag{2.2.5}$$

This fact, similarly justifiable for any arbitrary space-time finite element problem, allows to formulate the continuous and discontinuous time-stepping procedures from FET covered in the subsequent sections of the report. Moving from the variational problem on the Sobolev space W to the Galerkin approximating space W_{kh}, the full variational problem is to find $u_{kh} \in W_{kh}$ such that $\forall v_{kh} \in W_{kh}$, equation (2.2.5) holds for each of the time-steps t_n. We can now proceed with the substitution $t = t_n + \tau \Delta t$ in which $f(t) = \tilde{f}(\tau)$ such that $dt = \Delta t d\tau$, to pull-back the integral on the single time-step to the reference normalised interval $[0, 1]$:

$$\begin{aligned} &\int_0^1 \int_\Omega \Delta t \left(\tilde{v}_{kh} \frac{\partial \tau}{\partial t} \frac{\partial \tilde{u}_{kh}}{\partial \tau} + \nabla \tilde{u}_{kh} \cdot \nabla \tilde{v}_{kh} \right) dx d\tau = 0 \\ \implies \mathcal{I} := &\int_0^1 \int_\Omega \left(\tilde{v}_{kh} \frac{\partial \tilde{u}_{kh}}{\partial \tau} + \Delta t \nabla \tilde{u}_{kh} \cdot \nabla \tilde{v}_{kh} \right) dx d\tau = 0 \end{aligned} \tag{2.2.6}$$

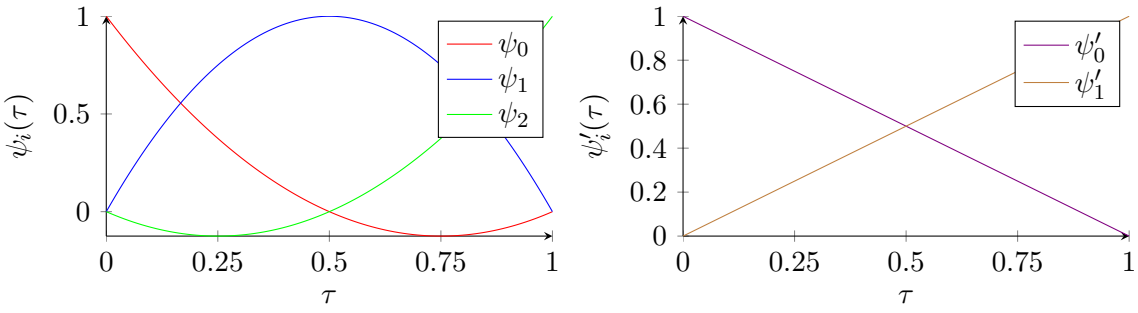
This is the space-time weak variational form for the homogenous heat equation problem over a single time-step, the starting point for the derivation of a FET time-stepping procedure.

2.2.3 Continuous Petrov-Galerkin Elements

Once a space-time variational problem has been formulated, the finite element spaces D_k and S_h have to be chosen for a solution to be sought in practice. First, it is noted that, by the tensor product space nature of W_{kh}, S_h can in principle be any spatial domain finite element space, therefore it is taken as a given for the problems considered in this report. To choose D_k, we begin by imposing the simplest requirement on the function space, namely that a function over a time mesh must be continuous between time elements. As discussed by Ern and Guermond in [15], continuous Petrov-Galerkin (cPG) elements satisfy this continuity requirement and decouple the FET problem into local problems over single time elements. This makes the FET timestepping that this investigation seeks possible to implement.

The cPG approach consists in picking spaces that are different for the trial and test functions that appear in the weak variational space-time problem. Keeping S_h homogenous between trial and test spaces, the time dependence of u_{kh} will derive from D_k spanned by the basis Ψ but the time dependence of v_{kh} will come from D_k' spanned by Ψ'. For D_k and D_k' being polynomial spaces, the cPG method requires the trial function space to be that of piecewise-continuous polynomials of degree p and the test function space to be of degree $p - 1$. Figure 2.4 depicts the trial and test basis functions across a reference unit time interval.

Let $k = \Delta t$ be the typical time mesh size and u_{kh} the solution to the weak variational problem on W_{kh} of interest. Let u be the exact solution of the weak variational problem on



(a) Trial function space quadratic basis functions (b) Test function space linear basis functions

Figure 2.4: Continuous Petrov-Galerkin trial and test function space basis functions for quadratic Lagrange finite elements

the Sobolev space W. Define the L^2 square integrated error over the domain $\Omega \times [0, T]$ as:

$$||u - u_{kh}||_{L^2}^2 = \int_0^T \int_\Omega |u(x,t) - u_{kh}(x,t)|^2 dx dt \tag{2.2.7}$$

Then [15] also proves that $||u - u_{kh}||_{L^2}$ scales as $\mathcal{O}(h^m + \Delta t^{p+1})$, with h being the usual spatial typical size and m being some constant. This means that the error for a FET discretisation scales as one degree higher than the chosen approximating polynomial degree.

Finally, [15] also establishes the equivalence of the cPG method with the Kuntzmann-Butcher implicit Runge-Kutta (IRK) family of timestepping methods, discussed in more detail in section 2.2.8. The lowest order cPG scheme, with linear trial functions and constant test functions, is the lowest order Kuntzmann-Butcher scheme: Crank-Nicolson [18].

2.2.4 Discontinuous Galerkin Elements

It is also possible to obtain a timestepping FET implementation by relaxing the constraint that trial functions, and therefore solutions, should be continuous across elements. This is known as the discontinuous Galerkin (dG) FET scheme. It presents important benefits in the stability of numerical solutions, due to the larger number of degrees of freedom available for polynomial fixing within elements. For the kinds of problems where continuity provokes large oscillatory behaviour of numerical solutions close to derivative discontinuities or shocks, this is fundamental to produce meaningful solutions which converge as $\Delta t \to 0$.

In the case of dG, the time Galerkin-approximating space D_k is the same for both trial and test functions and can be defined as the span of the basis Ψ, composed of discontinuous degree p polynomials when restricted to elements. Discontinuous elements require a change

to the specification of a space-time weak variational problem. Consider the following:

$$\int_0^T \int_\Omega \frac{\partial u}{\partial t} v dx dt = \sum_n \int_{I_n} \int_\Omega \frac{\partial u}{\partial t} v dx dt \tag{2.2.8}$$

$$= \sum_n \left[\int_{I_n} \int_\Omega \frac{\partial}{\partial t}(uv) dx dt - \int_{I_n} \int_\Omega u \frac{\partial v}{\partial t} dx dt \right] \tag{2.2.9}$$

$$= \sum_n \left[\left[\int_\Omega uv dx \right]_{\partial I_n} - \int_{I_n} \int_\Omega u \frac{\partial v}{\partial t} dx dt \right] \tag{2.2.10}$$

$$= \sum_n \left[\int_\Omega uv dx \right]_{\partial I_n} - \int_0^T \int_\Omega u \frac{\partial v}{\partial t} dx dt \tag{2.2.11}$$

In this case, the sum of terms over the endpoints of each sub element $I_n \in [0, T]$ does not telescopically reduce to the contributions from the boundary of $[0, T]$. This is because of the discontinuity of the functions across element boundaries. Moreover, the value on the boundary is multiply defined. Therefore, we introduce a numerical flux function $\hat{f}(u)(\mathbf{x}, t) = \hat{f}(u(\mathbf{x}, t^-), u(\mathbf{x}, t^+))$ [19] and impose that equation (2.2.11) is equal to:

$$\sum_n \left[\int_\Omega \hat{f}(u)(\mathbf{x}, t) v(\mathbf{x}, t) dx \right]_{\partial I_n} - \int_0^T \int_\Omega u \frac{\partial v}{\partial t} dx dt \tag{2.2.12}$$

There are many kinds of application-specific flux functions. For time elements, a valid numerical flux function is the standard upwind flux: the value of a function on a boundary between time elements is chosen to be the function value given by the previous element:

$$\hat{f}(u)(\mathbf{x}, t) = \hat{f}(u(\mathbf{x}, t^-), u(\mathbf{x}, t^+)) = u(\mathbf{x}, t^-) \tag{2.2.13}$$

This is consistent with the "arrow of time" assumption that a future evolution of a system does not affect its present and is well defined for each time element except for the first in a domain. An initial condition over a domain is therefore imposed weakly, by introducing it through the flux function $\hat{f}$ as the value passed by the "previous element" for the first time element.

Finally, [15] establishes that time dG has the same $\mathcal{O}(h^m + \Delta t^{p+1})$ space-time convergence properties of the L^2 error $||u - u_{kh}||_{L^2}$ of equation (2.2.7) as using cPG elements and that its schemes are equivalent to Radau IIA implicit Runge-Kutta.

2.2.5 Deriving the Heat Equation FET Block Systems for One Timestep

Consider the weak variational form for the heat equation over the first reference unit interval as in equation (2.2.6). In order to discretise it, continuous Petrov-Galerkin elements are chosen for a solution in time that is a degree p polynomial. The method requires the trial function basis $\Psi = \{\psi_i(t)\}_{i=0}^{p}$ and a one-degree-lower test function basis $\Psi' = \{\psi'_i(t)\}_{i=0}^{p-1}$. The spatial component remains equal for both space-time trial and test functions, meaning that the exclusively-spatial restriction of both functions at a single point in time is spanned by the $d+1$ dimensional basis $\Phi = \{\phi_j(\mathbf{x})\}_{j=0}^{d}$.

When restricted to this cPG finite element space, the expansions of the trial function u_{kh} and test function v_{kh} in the two bases are $u(\mathbf{x},t) = u_{ij}\phi_{ij}(\mathbf{x},t) = u_{ij}\tilde{\phi}_{ij}(\mathbf{x},\tau)$ and $v(\mathbf{x},t) = v_{ij}\phi'_{ij}(\mathbf{x},t) = v_{ij}\tilde{\phi'}_{ij}(\mathbf{x},\tau)$ with $u_{ij}, v_{ij} \in \mathbb{R}$ constants. Note that the Einstein summation convention of summing over repeating indices is again used, together with the tilde to represent the rewritten dependence on τ. Considering that the expansion coefficients and spatial basis functions of u_{kh} and v_{kh} are time-independent we have that:

$$\frac{\partial u_{kh}}{\partial \tau} = u_{ij}\frac{\partial \tilde{\psi}_i}{\partial \tau}(\tau)\phi_j(\mathbf{x}) \qquad \frac{\partial v_{kh}}{\partial \tau} = v_{ij}\frac{\partial \tilde{\psi'}_i}{\partial \tau}(\tau)\phi_j(\mathbf{x}) \tag{2.2.14}$$

We can now expand these expressions inside the variational form integral recalling that $t = t_n + \tau\Delta t$ to obtain:

$$\mathcal{I} = \int_0^1 \int_\Omega \left[u_{ij}\frac{\partial \tau}{\partial t}\frac{\partial \tilde{\psi}_i}{\partial \tau}\phi_j v_{kl}\tilde{\psi'}_k\phi_l + (u_{ij}\tilde{\psi}_i\nabla\phi_j)\cdot(v_{kl}\tilde{\psi'}_k\nabla\phi_l)\right] dx\frac{\partial t}{\partial \tau}d\tau \tag{2.2.15}$$

$$= \int_0^1 \int_\Omega \left[u_{ij}\frac{\partial \tilde{\psi}_i}{\partial \tau}\phi_j v_{kl}\tilde{\psi'}_k\phi_l + \Delta t(u_{ij}\tilde{\psi}_i\nabla\phi_j)\cdot(v_{kl}\tilde{\psi'}_k\nabla\phi_l)\right] dxd\tau = 0 \tag{2.2.16}$$

To computationally approximate the time interval, we perform numerical quadrature using a q point Gauss-Legendre quadrature rule of the following form:

$$\int_0^1 \int_\Omega F(\mathbf{x},\tau)dxd\tau \approx \sum_{m=1}^{q} b_m \left(\int_\Omega F(\mathbf{x},\tau_m)dx \right) = b_m \int_\Omega F(\mathbf{x},\tau_m)dx \tag{2.2.17}$$

Applying quadrature from (2.2.17) to the integral $\mathcal{I}$:

$$\mathcal{I} = b_m \int_\Omega \left[u_{ij}v_{kl}\frac{\partial \tilde{\psi}_i}{\partial \tau}|_{c_m}\tilde{\psi'}_k|_{c_m}\phi_j\phi_l + u_{ij}v_{kl}\tilde{\psi}_i|_{c_m}\tilde{\psi'}_k|_{c_m}\nabla\phi_j\cdot\nabla\phi_l\Delta t\right] dx \tag{2.2.18}$$

$$= b_m u_{ij}v_{kl}\frac{\partial \tilde{\psi}_i}{\partial \tau}|_{c_m}\tilde{\psi'}_k|_{c_m}\int_\Omega \phi_j\phi_l dx + b_m u_{ij}v_{kl}\tilde{\psi}_i|_{c_m}\tilde{\psi'}_k|_{c_m}\int_\Omega \nabla\phi_j\cdot\nabla\phi_l dx\Delta t \tag{2.2.19}$$

To simplify the obtained expression, it is useful to define the following matrices:

- Petrov time mass: $P^M_{ik} = b_m\tilde{\psi}_i|_{c_m}\tilde{\psi'}_k|_{c_m}$
- Petrov time half-stiffness on trial function: $P^L_{ik} = b_m\frac{\partial \tilde{\psi}_i}{\partial \tau}|_{c_m}\tilde{\psi'}_k|_{c_m}$

- Spatial mass matrix: $M_{jl} = \int_\Omega \phi_j \phi_l dx$
- Spatial stiffness matrix: $K_{jl} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_l dx$

Other quadrature dependent matrices that appear in other linear problems are the Petrov time half-stiffness on test function $P^R_{ik} = b_m \tilde{\psi}_i|_{c_m} \frac{\partial \tilde{\psi}'_k}{\partial \tau}|_{c_m}$ and time stiffness matrices $P^K_{ik} = b_m \frac{\partial \tilde{\psi}_i}{\partial \tau}|_{c_m} \frac{\partial \tilde{\psi}'_k}{\partial \tau}|_{c_m}$. D_{ik} represents these same matrices for time discontinuous Galerkin elements and basis functions. Using the defined matrix components, the spacetime variational form for the heat equation reduces (including summation notation for clarity) to the expression:

$$\sum_{i=0}^{p}\sum_{j=0}^{d}\sum_{k=0}^{p-1}\sum_{l=0}^{d} u_{ij} v_{kl} P^L_{ik} M_{jl} + \Delta t \sum_{i=0}^{p}\sum_{j=0}^{d}\sum_{k=0}^{p-1}\sum_{l=0}^{d} u_{ij} v_{kl} P^M_{ik} K_{jl} = 0 \tag{2.2.20}$$

From the variational problem in (2.2.5), equation (2.2.20) must hold for any v, therefore it must also hold for v specified in its basis expansion by the coefficients $v_{kl} = \delta_{ks}\delta_{lr}$ for $0 \le k, s \le p-1$ and $0 \le l, r \le d$ with δ being the Kroenecker symbol. This is equivalent to saying that the variational form must hold for each of the test space's basis functions separately. Hence, we retrieve an equation for each s, r pair:

$$\left(P^L_{is} M_{jr} + \Delta t P^M_{is} K_{jr}\right) u_{ij} = 0 \tag{2.2.21}$$

$$\implies G^{(i,s)}_{rj} u_{ij} = 0 \quad \text{with} \quad G^{(n,s)}_{rj} = P^L_{ns} M_{rj} + \Delta t P^M_{ns} K_{rj} \tag{2.2.22}$$

From the summation on the coefficients of u we extract a matrix block system, that is a matrix system where the entries are submatrices:

$$\underbrace{\begin{bmatrix} G^{(0,0)} & G^{(1,0)} & G^{(2,0)} & \dots & G^{(p,0)} \\ G^{(0,1)} & G^{(1,1)} & G^{(2,1)} & \dots & G^{(p,1)} \\ G^{(0,2)} & G^{(1,2)} & G^{(2,2)} & \dots & G^{(p,2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G^{(0,p-1)} & G^{(1,p-1)} & G^{(2,p-1)} & \dots & G^{(p,p-1)} \end{bmatrix}}_{\mathbb{R}^{p(d+1)\times(p+1)(d+1)}} \underbrace{\begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_p \end{bmatrix}}_{\mathbb{R}^{(p+1)(d+1)}} = \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}}_{\mathbb{R}^{p(d+1)}} \tag{2.2.23}$$

System (2.2.23) is underdetermined, since it seeks $(p+1)(d+1)$ unknowns from $p(d+1)$ equations. Considering the first time-step, the initial condition $u(\mathbf{x}, 0) = \rho(\mathbf{x})$ can be interpolated into the space S_h as $\rho_j \phi_j(\mathbf{x})$. This gives the condition $\forall j, 0 \le j \le d, u_{0j} = \rho_j$, yielding the required $d+1$ equations to make the block matrix square and (in general) invertible. So, the block system for the basis expansion coefficients of the solution for a whole timestep, assuming a continuous Petrov-Galerkin discretisation with a fixed initial condition is:

$$\begin{bmatrix} I & 0 & 0 & \dots & 0 \\ G^{(0,0)} & G^{(1,0)} & G^{(2,0)} & \dots & G^{(p,0)} \\ G^{(0,1)} & G^{(1,1)} & G^{(2,1)} & \dots & G^{(p,1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G^{(0,p-1)} & G^{(1,p-1)} & G^{(2,p-1)} & \dots & G^{(p,p-1)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{\rho} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \tag{2.2.24}$$

Where $\boldsymbol{\rho}$ is the vector of coefficients of the initial condition. It is worth noting that, like the time integral in (2.2.18), the integrals within M_{ij} and N_{ij} must be computed through numerical quadrature, as a symbolic computable solution is not guaranteed and oftencases unattainable. Once this system has been setup after quadrature, a solver such as PETSc [8] can be used solve for the coefficients of the basis expansion of the approximated solution.

2.2.6 Extension to More Time-Steps

Having obtained a matrix block system for the first time-step $t_0 < t < t_1$, the immediate consequence is to iterate the cPG approach with the following time-steps, enforcing continuity by matching the final function value over a timestep to the initial condition of the following step. Consider the case of lowest order Petrov-Galerkin, composed of linear test functions and constant trial functions. In this case, each element has no intermediate function values. Then, for an initial condition coefficient vector ρ, the block matrix systems for the first two timesteps to be solved are:

$$\begin{bmatrix} I & 0 \\ G^{(0,0)} & G^{(1,0)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\rho} \\ \mathbf{0} \end{bmatrix} \qquad \begin{bmatrix} I & 0 \\ G^{(0,0)} & G^{(1,0)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{0} \end{bmatrix} \tag{2.2.25}$$

This procedure can be repeated for all intervals of the global time mesh P^N, propagating up to $t_{N-1} < t < t_N$ to give the full space-time approximation of the solution $u(\mathbf{x}, t) = u_{ij}\phi_{ij}(\mathbf{x}, t)$. Each of the block systems can be combined into the bigger block system:

$$\begin{bmatrix} I & 0 & 0 & 0 & \cdots & 0 & 0 \\ G^{(0,0)} & G^{(1,0)} & 0 & 0 & \cdots & 0 & 0 \\ 0 & G^{(0,0)} & G^{(1,0)} & 0 & \cdots & 0 & 0 \\ 0 & 0 & G^{(0,0)} & G^{(1,0)} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & G^{(1,0)} & 0 \\ 0 & 0 & 0 & 0 & \cdots & G^{(0,0)} & G^{(1,0)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_{n-1} \\ \mathbf{u}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{\rho} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{2.2.26}$$

In (2.2.26) we have $I \in \mathbb{R}^{(d+1)\times(d+1)}$, $0 \in \mathbb{R}^{(d+1)\times(d+1)}$, $\boldsymbol{\rho} \in \mathbb{R}^{d+1}$ is the vector of coefficients of the projection of initial condition $\rho(\mathbf{x})$ and $\mathbf{0} \in \mathbb{R}^{d+1}$ is the $d+1$-dimensional zero vector. The system can be solved either by a mixture of forward substitution and inversion on rows, exploiting the visibly banded structure of the matrix, or more practically monolithically, by applying LU patch preconditioners to solve the system in parallel. The structure of the block system also shows how the problem has been decoupled over timesteps.

This approach of forming a larger block system that combines multiple timesteps can be generalised trivially for time elements that are of a polynomial degree greater than 1.

2.2.7 Nonlinear FET and Burgers' Equation

The homogenous heat equation is a linear PDE, that is if u_0, u_1 both solve the heat equation problem then $u_0 + u_1$ also solves it. This is what allows composition of the bilinear forms in

the test and trial functions that lead to the block matrix system of equation (2.2.24). Nevertheless, many finite element problems of great interest are nonlinear in the trial function (they must all be linear in the test function by construction of the variational forms). This implies that no usual matrix-vector system can be produced to obtain a solution. As the implementation presented in this investigation supports nonlinear problems, it is important to outline the general solution procedure for nonlinear systems.

Consider the functional $f(u;v)$, where the semicolon indicates that f is linear in all the arguments that follow it but possibly nonlinear in those that precede it. Then f is called a residual for a given weak variational problem if $f(u;v) = 0$ for v arbitrary if and only if u solves the variational problem and satisfies its boundary conditions. For the linear Poisson problem of (2.1.6), for example, the residual can be defined as:

$$f(u;v) = b(v,u) - L(v), \qquad f(u;v) = 0 \iff b(v,u) = L(v) \tag{2.2.27}$$

The requirement that $f(u;v) = 0$ for v arbitrary implies that iterative root finding algorithms can be used to approximately solve the residual problem, assuming they are modified to seek functions instead of scalars. The most common example of such an algorithm is the Newton iterative scheme [20]. As detailed in [20], one needs to supply an initial guess u^0 for the solution, after which linearisation of the residual with respect to a perturbation function is obtained using the Gateaux derivative and an updated solution guess is computed. The algorithm halts when an update meets convergence criteria or when the procedure is deemed non-convergent, for which a better guess is to be supplied.

As Fetsome exploits Firedrake's standard solution layer built on PETSc for nonlinear problems, for more specific detail the reader is referred to [20].

To combine FET and iterative nonlinear solution procedures, it suffices to use FET to obtain a residual form that is linear in its test function v. To illustrate how this can be done, consider the one-dimensional nonlinear Burgers equation over a timestep $\Omega \times I_n$ in conservative form:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial}{\partial x}(u^2) = 0 \tag{2.2.28}$$

$$u(x,0) = u_0(x) \tag{2.2.29}$$

We can obtain the weak variational problem and residual by picking a space-time finite element space with time trial basis Ψ, time test basis Ψ' and piecewise-continuous space basis Φ. We perform the standard multiplication and integration procedure:

$$f(u;v) = \int_{I_n}\int_\Omega \left(v\frac{\partial u}{\partial t} + \frac{1}{2}v\frac{\partial}{\partial x}(u^2)\right) dxdt \tag{2.2.30}$$

$$= \int_0^1 \int_\Omega \left(\tilde{v}\frac{\partial \tilde{u}}{\partial \tau} + \Delta t\frac{1}{2}\tilde{v}\frac{\partial}{\partial x}(\tilde{u}^2)\right) dxd\tau \tag{2.2.31}$$

$$= \int_\Omega \left[\int_0^1 \sum_i v_i\tilde{\psi}_i' \sum_j u_j \frac{\partial \tilde{\psi}_j}{\partial \tau} d\tau + \Delta t \int_0^1 \sum_i \frac{1}{2} v_i \tilde{\psi}_i' \frac{\partial}{\partial x}\Big(\sum_j u_j\tilde{\psi}_j\Big)^2 d\tau\right] dx \tag{2.2.32}$$

In the above, $v_i = v_i(x) = \sum_k v_{ik}\phi_k(x)$ and $u_j = u_j(x) = \sum_k u_{jk}\phi_k(x)$. As for the linear heat equation, we introduce a quadrature rule to approximate the time integral.

$$\implies f(\mathbf{u};\mathbf{v}) = \sum_i \sum_k v_{ik} \int_\Omega \phi_k \left[\sum_m b_m \Big(\tilde{\psi}_i'|_{\tau_m} \sum_j \tilde{\psi}_j|_{\tau_m} u_j + \tilde{\psi}_i'|_{\tau_m} \frac{\partial}{\partial x} \Big(\sum_j \tilde{\psi}_j|_{\tau_m} u_j \Big) \Big) \right] dx \tag{2.2.33}$$

Where $f(u;v) = f(\mathbf{u};\mathbf{v})$ to indicate the rewritten dependence of the residual on the coefficient vectors that express u and v in their bases. By specification of the trial and test spaces, the values of $\tilde{\psi}_i'(\tau_m)$ and $\tilde{\psi}_j(\tau_m)$ are both known at all the points τ_m. Therefore, we have obtained a problem which is linear in the test function and unknown exclusively in the spatial components of the integral. So, an iterative procedure as the one already supported by Firedrake can be used to obtain a solution automatically.

2.2.8 FET and Runge-Kutta Time-Stepping Schemes

Runge-Kutta time-stepping schemes are very common methods to numerically approximate solutions to time-dependent ordinary and partial differential equations. The stepping schemes they use can differ both in implementation and the properties they guarantee, so Runge-Kutta integrators constitute a large family of numerical methods.

Let the following be an ODE for u over a time domain $[0,T]$, $f : (0,T] \times \mathbb{R} \to \mathbb{R}$, for which the solution is sought together with its initial condition:

$$\frac{du}{dt} = f(t,u), \quad \text{where} \quad u|_{t=0} = \rho, \rho \in \mathbb{R} \tag{2.2.34}$$

To obtain a Runge-Kutta approximation scheme, define the weights b_i, the abscissae c_i and the coefficients a_{ij}, where $1 \leq i,j \leq s$ and s is the chosen number of stages. Let $D = \{0 = t_0 < t_1 < \cdots < t_n = T\}$ be a finite partition of the time domain $[0,T]$ and define $h_n = t_{n+1} - t_n$ as the size of a single time-step. Let u^n denote the approximated solution at time t_n. We consider an s-stage Runge-Kutta solution to the ODE in equation (2.2.34) to be defined as:

$$u^{n+1} = u^n + h_n \sum_{i=1}^{s} b_i k_i \tag{2.2.35}$$

$$k_i = f(t_n + c_i h_n, u^n + h_n \sum_{j=1}^{s} a_{ij} k_j) \tag{2.2.36}$$

The formulation generalises to PDEs through the method of lines or Rothe's method (see 2.3). Runge-Kutta schemes therefore approximate the evolution to u^{n+1} from u^n through a time-step by a weighted average of gradients evaluated at different sample points around the known point. It is worth noting that the expression for the i^{th} stage, k_i, can depend on itself from its inclusion in the arguments to f based on the coefficients a_{ij}. This differentiates what are called implicit methods, that do present such inclusions, from explicit methods, where no expression for a stage k_i can include itself on the right-hand side.

As discussed in sections 2.2.3 and 2.2.4, a choice of continuous Petrov-Galerkin elements in time is equivalent to using Kuntzmann-Butcher IRK schemes and a choice of discontinuous Galerkin elements is equivalent to using Radau IIA IRK. The numerical analysis of implicit Runge-Kutta methods therefore can be applied to obtain important properties of the two FET discretisations considered. As discussed in [21], Kuntzmann-Butcher (or Gauss) IRK has properties desirable for solving PDEs of physical interest. KB IRK is A and B stable, as well as symplectic. On the other hand, Radau IIA IRK is important in the solution of stiff problems through its L stability [5], although it does not possess the same symplecticity properties as KB IRK. For more detail the reader is referred to [21].

Besides the properties of FET obtained by its correspondence to certain Runge-Kutta methods, there are advantages that a pure FET implementation has over relying on equivalent timestepping schemes. First of all, many of the intermediate "stage" values produced by traditional time stepping procedures cannot be interpreted as solutions of the problem within a time step, so solutions are fundamentally discrete. FET guarantees that the intermediate

solution values are valid in approximating a solution within a time step. Secondly, the freedom given by FET in choosing the polynomial degree of time finite elements implies that higher order equivalent time stepping methods are automatically derived in a unified way. This makes development of higher order schemes easier and more straightforward. Moreover, equivalent known time stepping methods often rely on integrating the time domain with exact quadrature. The arbitrary choice of quadrature in FET allows under-integrated time stepping schemes to also be obtained, sacrificing a loss of accuracy to obtain systems which are still meaningful but more easily solved computationally, through parallelisation for example. Finally, it can be argued that a unified approach between space and time discretisations can yield a more natural formulation of numerical solutions to time-dependent PDEs.

2.3 Current State of the Art for Time PDEs

One of the goals of Fetsome is to implement FET functionality that closely follows the mathematical abstractions that underpin it. This section evaluates this on the current state of the art in solving time dependent PDEs using the finite element method, with a focus on packages that present some space-time-like solvers.

The two most common methods found in modern FEM packages for solving spatially dependent systems are the method of lines and Rothe's method [15]. The method of lines discretises the space domain using finite elements to obtain a system of coupled time domain ODEs, which can subsequently be solved using Runge-Kutta timestepping, for example. Rothe's method discretises in the time domain first, producing weak forms that are then solved with the finite element method iteratively. The two methods are equivalent for many applications.

The FEniCS framework is closely related to Firedrake, providing a similar level of mathematical abstraction through its use of UFL, which it maintains. Its natural UFL support means that both the method of lines and Rothe's method can be used to solve time dependent problems in FEniCS. Unicorn [22] is a FEniCS fork that uses space-time finite elements to solve problems in continuum mechanics. Its implementation is restricted to the adaptive G2 (General Galerkin) discretisations and currently only implements the cG(1)cG(1) space-time method, which is equivalent to Crank-Nicolson timestepping (or first order continuous Petrov-Galerkin). This is due to the package's primary focus on moving space-time element meshes along PDE characteristics rather than implementing FET in its generality.

The deal.II library is a well-known and important high performance C++ finite element library. Despite not exploiting a DSL like UFL, its object oriented approach to the formulation of weak variational problems is compatible with both the method of lines and Rothe's method for solving time-dependent PDEs, as shown in [23]. Another particular method supported by deal.II of solving a type of such PDEs is through Lie and Strang splitting [24]. Nevertheless, deal.II does not natively support FET schemes. Furthermore, an advantage of Firedrake over deal.II is its sophisticated code generation framework, which automatically makes high-performance solving possible.

Beyond FEnICS and deal.II, other software packages such as DUNE [4] and Hermes [25]

were found to focus on the method of lines and Rothe's method for the solution of time-dependent PDEs. They did not present any FET functionality.

It can be concluded that the implementation of FET timestepping into a generic FE software framework is a relatively unexplored task. Although its abstract mathematical theory is well established, a generic formulation for a FET computational abstraction and its integration with existing software presents a novel and interesting opportunity to contribute to the landscape of these automated solver packages.

2.4 Implementation Background

2.4.1 UFL (Unified Form Language)

UFL [6] (Unified Form Language) is a domain specific language that allows users to describe finite element problems posed on spatial domains in code. It is developed and maintained as part of the FEniCS Project and is released as a freely available Python library. It is integrated within Firedrake to serve as a backbone for the specification of finite element problems as seen by the users.

UFL is rich in its functionality: it gives the user the ability to specify variational forms of weak PDEs distinguishing trial, test and forcing functions from each other, it integrates function spaces (standard and mixed) with meshes and presents a variety of reference elements on which to pose finite element problems. In its lower levels, UFL also presents a library to access the internal representation of expressions and forms as well as algorithms to manipulate them, favouring a high level of automation in the applications that use it. In the case of the FET implementation presented in this study, UFL is central in the symbolic manipulation layer that handles the first time-integration layer in mixed form assembly. In the subsequent stages, Firedrake makes use of UFL to assemble the matrix-vector systems that are to be solved in its implementation of the finite element method. Using UFL is covered in greater detail in section 3.2.

2.4.2 Irksome

Irksome [5] is a high-level library that is built on top of Firedrake to target a more automatic solution of time-dependent PDEs. It provides abstractions that extend UFL to represent time derivatives inside variational forms and implements Runge-Kutta timestepping procedures in a unified, user-centred manner. Irksome uses UFL's form manipulation capabilities to relieve a user from manually describing weak variational forms for each intermediate and final-step stage of an RK procedure. A user can choose a specific method's Butcher tableau and supply it to one of the available time steppers which handle advancing the solution as well as enforce initial conditions, boundary conditions and pass efficiency-related parameters to the lower level solvers. This allows a great variety in the choice of timestepping methods for users only requiring mathematical knowledge, as well as increasing the flexibility a user has on changing the order of the chosen scheme or changing the chosen RK scheme entirely almost effortlessly. The following is an example of the Irksome specification of the semidiscrete UFL form for solving the heat equation:

```
F = inner(Dt(u), v) * dx + inner(grad(u), grad(v)) * dx
```

Rather than targeting arbitrary RK methods, Irksome focuses on implicit RK (IRK) schemes. Through their Butcher tableau specification, Irksome produces discrete algebraic systems based on the weak variational problem of interest that exploit Firedrake's solver infrastructure and efficiency optimisations to solve the time dependent problem. IRK methods are important for PDEs of physical interest specifically, since their numerical structures provide desirable stability and integral conservation properties. Some IRK methods that Irksome currently supports include Lobatto IIA, Lobatto IIC, Radau IIA and Qin-Zhang.

By the equivalence of dG in time with Radau IIA methods, Irksome already supports a FET-equivalent method. Nevertheless, the much wider generality obtained by developing a full FET implementation that can be extended beyond time stepping procedures, as well as an implementation that gives more freedom on the precision of integration over the time domain, motivates Fetsome as a desirable extension of Irksome's current capabilities.

2.4.3 PETSc

PETSc [8] is the Portable Extensible Toolkit for Scientific computation. Through the Python library `petsc4py`, it forms the backbone of Firedrake's linear and nonlinear systems of equations solvers. PETSc provides a variety in the choice of solvers, as well as algebraic system preconditioning capabilities and MPI (Message Passing Interface)-supported parallelism, which makes it possible for larger scale problems to be run on supercomputers. Beyond the efficiency at solver-level, PETSc also targets the memory impact of large equation systems by providing compressed sparse row storage of matrices with potentially millions to billions of degrees of freedom. It also provides the DMPlex API to handle the mesh topology and unstructured mesh data needed for the assembly of global systems.

In Fetsome's implementation, PETSc's relevance hides behind the mixed function space formulation that leads to the large linear and nonlinear block systems that have to be solved (as discussed in [2]). This is handled most directly by the existing layers of Firedrake that support mixed function spaces and nonlinear solvers, not requiring any substantial modification on the side of the presented Fetsome implementation. This means that the scalability, efficiency and solver variety given by PETSc is effortlessly integrated into Fetsome. Finally, in line with the PETSc philosophy that solver choices should not be hard-wired and that the maximum flexibility in solver setup should be guaranteed, it is important for the Fetsome solution layer to allow passthrough of all solver options to the Firedrake solver layer.

2.4.4 FIAT and TSFC

FIAT [9] is the FInite element Automatic Tabulator. It supports arbitrary order finite elements in up to three dimensions as well as tabulation of equispaced or spectral variant basis functions on the elements. This means that FIAT also accounts for the automatic creation and evaluation of the arbitrary-order quadrature schemes used for numerical integral evaluation in Firedrake. Besides underpinning the one dimensional time elements used in this implemen-

tation, FIAT is effectively used by Fetsome for the creation of the quadrature schemes used in continuous Petrov-Galerkin and discontinuous Galerkin in time, as well as the evaluation of spacetime functions at single timepoints.

TSFC [11] is Firedrake's Two Stage Form Compiler which compiles UFL forms into efficiently scheduled tensor algebraic operations in C. GEM is a tensor algebra language that holds the intermediate tensor representation of forms for TSFC. The two stages of the compiler involve first managing the construction of algebraic objects from UFL to GEM and then the scheduling of the tensor operations to evaluate the lowered forms. TSFC and GEM are important for Fetsome as they provide memoising utilities for efficient UFL form expression tree traversal and transformations, exploited in the mixed function space time form generation layer.

2.5 Further Considerations

2.5.1 Ethical Considerations

Although this study and the interested implementation in themselves involve very minor ethical considerations, important observations must be made on the possible applications of the developed software and this study.

First of all, it is worth noting that neither the study, the implementation or the development of such implementation involve human participants. This is important because there are no ethical considerations to be made regarding personal data collection, reconstruction or diffusion. Similarly, there is no unethical behaviour that can be associated to user localisation. Also, this study makes no use of past personal data previously collected by third-party entities, as its purpose is to aid the development of purely mathematical software that does not involve datasets of any kind.

The most significant ethical concern tied to this study and the developed implementation is that of its potential of application in military scenarios. Finite element analysis is common in aviation resulting in its use, for example, to design an electromagnetic aircraft launcher for aircraft carriers, as shown by [26]. Similarly, studies like the one proposed by [27] demonstrate time dependent problems modelled through FEA for ballistic applications. In such a way, this study could be applied to improve the software actively employed in the development of military equipment or ammunition. It can nevertheless be discussed that the mathematical software developed can also have ethical impacts in these same fields. Better ballistic protection, the subject of [27], is fundamental for the role of civilians such as reporters or rescue volunteers in war zones.

Due to its wide applicability in the engineering industries, it is of no surprise that FEA for time-dependent problems has the potential for unethical environmental applications. We can understand from [28] the potential of application of this study to the improvement of cost effective oil piping for gas ducts and offshore rigs and in consequence to oil extraction methods, which can be seen as unethical behaviour given the impact of oil consumption on climate change. Similarly, finite element analysis has been used in fracking, as shown by [29], outlined as a greatly environmentally-problematic practice by [30] amidst its arguments against the method. Despite these examples, we can argue that improvement in this technology fundamentally leads to and increase in the safety of such processes. This can bring substantial advantages such as lowering the probability of an environmentally destructive oil spill or landslides as a result of fracking practices. Furthermore, FEA can also be used to drive positive environmental research, such as in [31], which discusses improvements to an electric aircraft engines to increase the efficiency of aerial vehicles.

One final ethical consideration that can derive from the automation of finite element in time solvers is the potential for unintentional misuse. It can be argued that fundamentally correct automation does in fact eliminate the possibility for human error in developing finite element software. Nevertheless, the reduced need for expertise might lead to misjudgement and misinterpreting of the results provided by this implementation, as suggested by Wheatley in [32]. It is suggested that any user should demonstrate care in interpreting the results that

the presented software obtains.

2.5.2 Legal Considerations

In addition to the ethical implications discussed above, some legal considerations tied to the implementation of the software are outlined. Firedrake is licensed under the GNU LGPL (Lesser General Public License), no code in the presented implementation impedes it from remaining under this license. In particular, no code under the GNU GPL (General Public License) was used as part of the implementation, as this would force Firedrake to forego its LGPL for GPL. Apart from this, the written implementation remains under the LGPL from its full integration with the Firedrake package.

Chapter 3

`Fetsome` Implementation

3.1 Additions to Irksome

Fetsome is the finite element in time implementation presented by this investigation. It is an extension to Firedrake that makes use of the Irksome package, in particular its UFL extensions to represent time derivatives. Fetsome allows Irksome users to choose FET timestepping as an alternative method to its implicit Runge-Kutta timestepping for solving time dependent partial differential equations. This section illustrates the main FET additions to the Irksome software base, explained in subsequent sections of Chapter 3.

3.1.1 Overview of Additions Functionality

The proposed implementation of finite element in time as an extension of Irksome consists in a variety of additions to its codebase.

The primary goal of such an implementation is to present the user with a way of specifying space-time domain finite element problems in a fashion compatible with the current Irksome and Firedrake practices. This means that the interface-driven objective of the implementation is to solve a FET problem starting from its weak variational form, specified in UFL forms, equipped with the finite element function spaces for the time and space domains. The first layer a user is therefore in contact with is that implemented by the `VariationalTimeStepper` class, which provides a time-stepper object a user can initialise with a space-time problem. This class can then be asked to evolve an initial condition according to the given problem by `advance`ing.

Directly below a `VariationalTimeStepper`, the infrastructure required to reduce a symbolic space-time variational problem to a mixed form problem solved by Firedrake is implemented by the `TimeFormGenerator`s. All implemented subclasses of such generator use varying combinations of trial and test function spaces over the time domain to assemble block systems equivalent to those presented in section 2.2.5. They translate purely symbolic space-time problems over space-time elements to mathematically equivalent multiple coupled spatial-only problems specified in UFL. This is done such that the mixed forms that result from the `TimeFormGenerator`s can be solved through standard Firedrake interfaces, ultimately reducing to computationally-intensive matrix-vector systems solved by the high-performance PETSc package.

In order to assemble space-time block systems from arbitrary forms automatically, the different kinds of `TimeFormGenerators` make extensive use of UFL expression and form tree traversers and transformers. The `split_time_orders_on` and `strip_dt` utilities perform the manipulation of symbolic space-time forms to reduce them to spatial-only forms separated in their degree of time derivatives. The `TimeQuadrature` class provides tabulation of time functions at arbitrary quadrature points, alongside the time mass, stiffness and half stiffness matrices that simplify form generation for linear problems. The `TimeFormGenerators` use `TimeQuadrature` to decompose time domain integrals into sums of weighted function evaluations at quadrature timepoints. For exact or high precision computation of integrals, quadrature degree of precision is estimated by traversing the integrals' expression trees by the `TimeDegreeEstimator` class and `estimate_time_degree` utility. This is then used to initialise the correct `TimeQuadrature` scheme.

More of the complicated mathematical capabilities of the finite element in time implementation reside in the form generation subclasses. These include, but are not limited to, variational problems which involve forcing functions, non-vanishing time boundary terms and time upwinding (see section 3.6.1). The `VariationalTimeStepper` class also acts as an interface for the user to access and make use of these in their formulation of weak variational space-time problems. Figure 3.1 includes the most important Fetsome additions and its interaction with the Firedrake and Irksome layers.

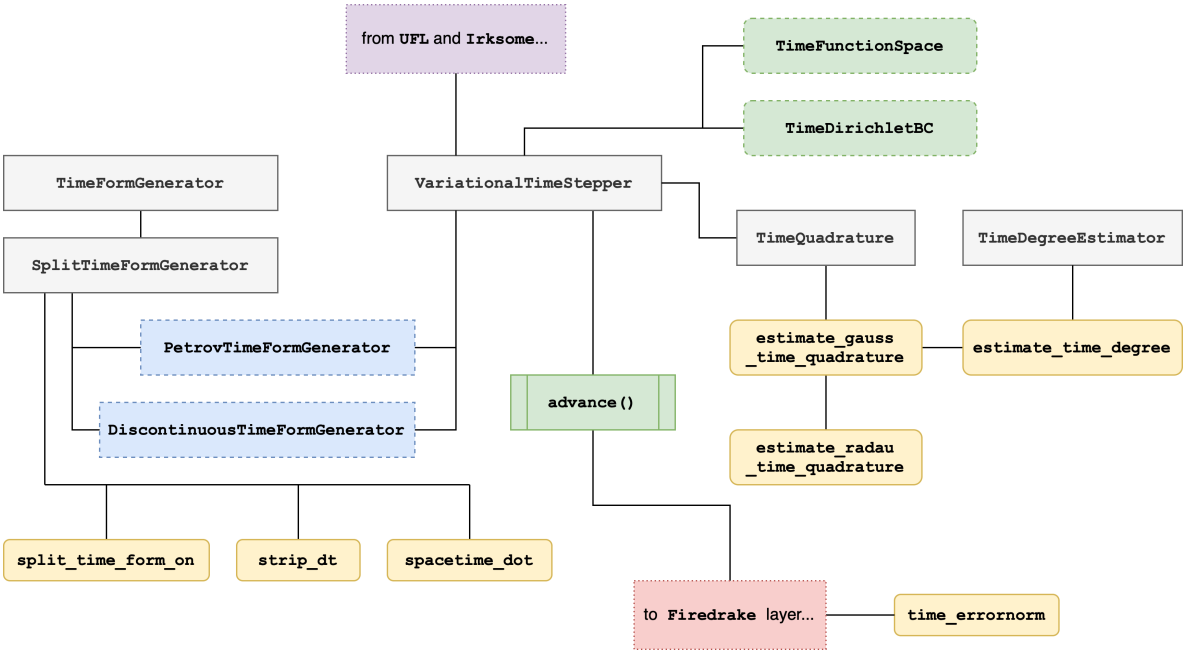


Figure 3.1: Map of the main classes and functions added in the Fetsome layer discussed in this implementation section

3.2 Using UFL

As Firedrake's and Irksome's symbolic manipulation layer is built on UFL, introduced in section 2.4.1, the finite element in time implementation subject of this study relies heavily on

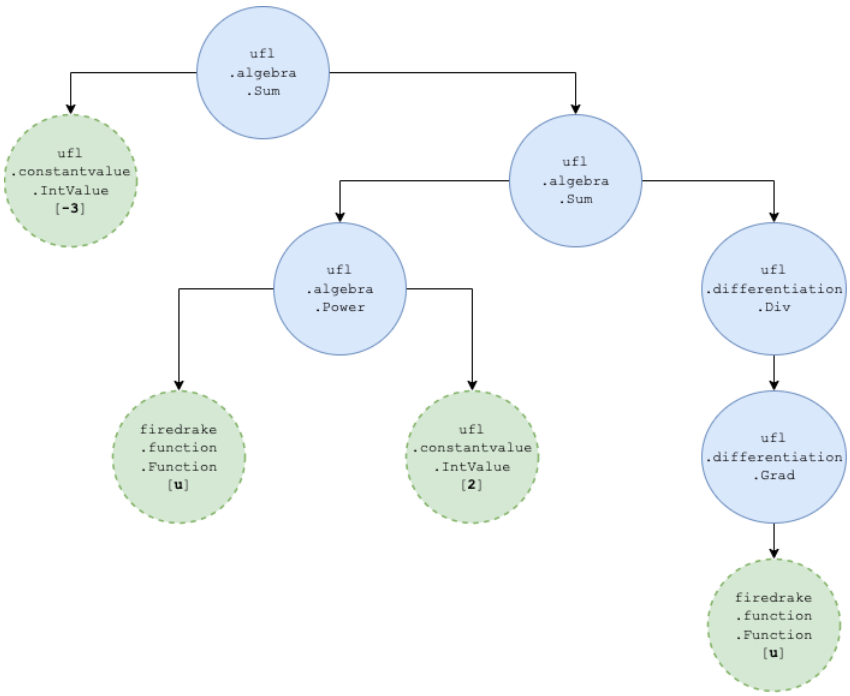


Figure 3.2: UFL expression tree for $\nabla^2 u + u^2 - 3$

its functionality for time form assembly. This section outlines the structure of UFL expressions and forms, as well as Irksome's time derivative extension to UFL, and the most important ways to manipulate them.

3.2.1 Internal Expression and Form Representations

The most basic layer in UFL's representation of weak variational problems consists of its expression representation layer. All expressions are represented through expression trees which implicitly handle operator precedence and arity, and are optimised when manipulated for evident opportunities such as products with zero, operator associativity and commutativity. UFL defines a hierarchy of classes that constitute its expressions.

The base class of all UFL expression types is the `ufl.core.expr.Expr` class. Common functionality that all expressions present includes (but is not limited to) `evaluate` for evaluation of expressions at coordinate points, `ufl_operands` to obtain tree-node sub-expressions and `dx` for automatic symbolic differentiation of the expression with respect to a spatial coordinate. Some fundamental UFL classes that are subclasses of `Expr` are the `Sum`, `Product` and `Division` classes (as part of the `ufl.algebra` package) or the `Dot`, `Inner` and `Outer` classes (as part of the `ufl.tensoralgebra` package). Figure 3.2 presents the expression tree for $\nabla^2 u + u^2 - 3$, encoded as the following UFL expression:

```
1 expr = div(grad(u)) + u**2 - 3
```

It is to be noted that `ufl_operands` returns an array instead of presenting a way to obtain the specific positional operands in an expression, for example the numerator and denominator in

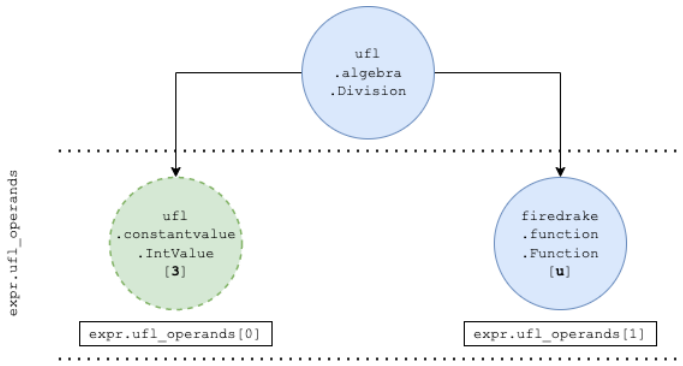


Figure 3.3: UFL expression tree for `expr = 3 / u` showing `ufl_operands` order

a division expression. In the case of commutative or symmetric operators this has no effect. For non-commutative operators (such as division), UFL follows the convention that operands are sorted by their appearance in the typed expression. For example, `expr = 3/u` has the expression tree depicted in figure 3.3. For a more comprehensive overview of UFL expression types and examples, the reader is referred to [6].

The form layer is the second layer in UFL's representation of weak variational problems. Forms in UFL are restricted to sums of integrals that can be taken on different subdomains of the global problem domain. The UFL class representing forms is `ufl.form.Form` and each separate integral within a form is an instance of `ufl.integral.Integral`. The simplest example of a linear form expressed in UFL is a single integral of a forcing function multiplied by a test function over the entire spatial domain: `F = f*u*dx`. The integral, where `f` is the forcing function and `v` is the test function, is represented by multiplication by a measure, in this case `dx`. Integrals in forms defined over meshes can also be taken over domain boundaries, represented by the measure `ds`, and interior facets between cells, with the measure `dS`. UFL gives more ways to access measure or domain information, most importantly through `ufl.integral.Integral.integral_type` and `ufl.integral.Integral.ufl_domain`. Figure 3.4 illustrates the internal structure of the forms used for the Helmholtz problem as specified in UFL within Firedrake, resulting from the following code:

```
b = dot(grad(u), grad(v))*dx + u*v*dx
F = f*v*dx
```

To replace sub-expressions within forms with new expressions, one can use the `ufl.replace` utility. The following example illustrates this:

```
F = dot(grad(u), grad(v)*dx + u*v*dx
Fnew = replace(F, {u: u_0})
# Fnew is dot(grad(u_0), grad(v)*dx + u_0*v*dx
```

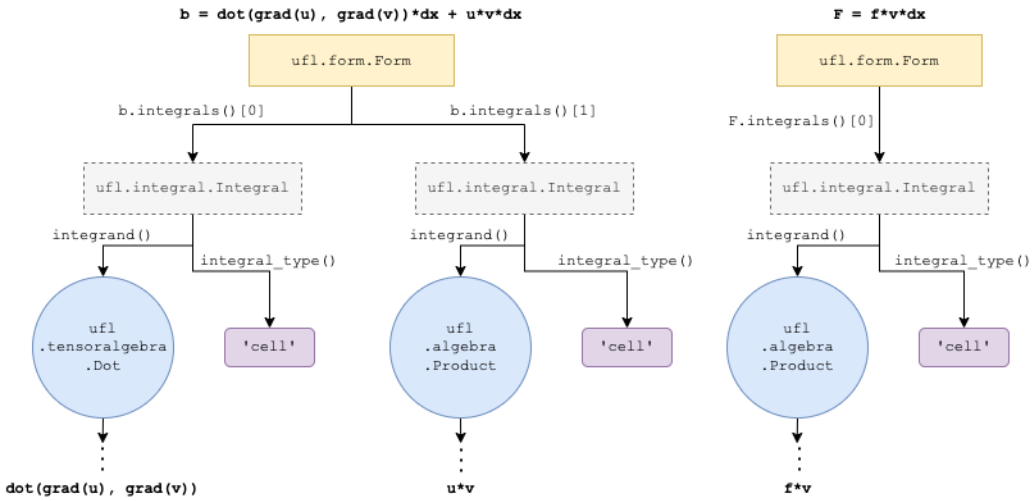


Figure 3.4: Internal structure of `b` and `F` forms for the Helmholtz problem

3.2.2 Irksome TimeDerivative and the Time Variable

Due to the spatial focus of UFL, time derivatives are not directly representable in its standard release. To allow manipulation of time derivatives, the `irksome.deriv.TimeDerivative` class and the `irksome.deriv.Dt` function are implemented by Irksome [5] to instantiate such a term in UFL expressions, and therefore also in forms. The following example illustrates the strong form of the heat equation as a UFL expression with Irksome's `Dt`:

```
heat = Dt(u) - div(grad(u))
```

All standard UFL functionality for expressions is available inside `TimeDerivative`s, since they are a subclass of `ufl.differentiation.Derivative` and therefore are UFL expressions.

Similarly, UFL does not have a dedicated time variable object. In Irksome, it is customary to declare `t` as a UFL `Constant`, so this is the representation that Fetsome also adopts.

3.3 Time Quadrature and Associated Objects

3.3.1 The `TimeQuadrature` Class

The `TimeQuadrature` class produces all the objects needed to handle the numerical computation of integrals in the time domain that are central to FET. As described in section 2.2.1, a quadrature rule is fundamentally composed of a set of quadrature points and the weight associated to each quadrature point. In the case of the time domain, integration is always one dimensional.

A quadrature object essentially only requires the quadrature points and weights that describe the numerical scheme. Once such elements are supplied, the `TimeQuadrature` class has the capability of composing time mass, stiffness and half stiffness matrices that can be used to solve linear problems (of section 2.2.5) by using the time finite element spaces that

support a form's trial and test functions. The class can also interact with a time finite element to evaluate its basis functions at the quadrature points, producing a vector rather than a matrix. This generalisation allows direct substitution of the time-basis expansion of functions that is used to solve nonlinear problems. As discussed, the trial and test function spaces in time need not be the same, such as in Petrov-Galerkin solutions for continuous-in-time solutions. Therefore, distinct trial and test function spaces for the time domain can be supplied to a quadrature object to generate the desired matrices. A `TimeQuadrature` object can be initialised with an arbitrary quadrature rule, although `make_gauss_time_quadrature` and `make_radau_time_quadrature` can be used to make Gauss and Gauss-Radau quadrature schemes for an arbitrary number of quadrature points. This automatic generation of specific quadrature schemes uses FIAT utilities to generate the points and weights for a `ufc_simplex` of dimension 1 (the UFC unit interval) and then builds `TimeQuadrature` objects compatible with the rest of the implementation.

3.3.2 Automating the Choice of Quadrature

Each quadrature scheme has its own degree of precision, that is the highest polynomial degree that can be exactly integrated by that scheme. For forms to be automatically integrated exactly, the maximum polynomial degree of a form can be calculated to then choose a quadrature scheme with the desired degree of precision. To do this, Fetsome uses a modification of UFL's `SumDegreeEstimator` class, which in standard UFL estimates the polynomial degree in space for an arbitrary expression. The `SumDegreeEstimator` is a `MultiFunction`, that is a UFL non-recursive node handler that traverses an expression tree. Since not all expressions can be integrated exactly (take for example a $\sin(x)$ curve), this existing estimator class makes use of heuristics that can be translated and reapplied easily in the time domain. Therefore, the main modification that the Fetsome `TimeDegreeEstimator` requires is that of ignoring the spatial polynomial degree of forms but handling time polynomial degree like a `SumDegreeEstimator` handles spatial degrees.

A `TimeDegreeEstimator` is supplied with a UFL `Constant` that represents the time variable `t` as well as the polynomial degree of the trial function space (heuristically always equal or greater than for the test function space). The most significant changes are made to the handling of constants, spatial coordinates, arguments, coefficients and spatial derivatives. For constants, it is retained that their polynomial degree is 0, with the modification that a check for the constant `t` must be made and handled as a polynomial of degree 1. For spatial and spatial-cell-coordinates, the time polynomial degree becomes 0. For arguments and coefficients (i.e. known and unknown functions), the polynomial degree is taken to be that passed to the estimator on initialisation. For spatial derivatives, a passthrough handler `_ignore_spatial_dx` is defined. Combinations of degrees in operations such as taking a sum of degrees for products or a maximum over degrees for sum operations is handled directly by the estimator's superclass.

Finally, the `TimeDegreeEstimator` also handles the newly available Irksome `TimeDerivative` UFL type by reducing the polynomial degree of the derivative operand.

Once the time degree estimator is applied as a `MultiFunction` to a form, integral or expression by `estimate_time_degree`, the calculated degree is used in time quadrature utilities

such as `estimate_gauss_time_quadrature` to calculate the number of points needed for an exact quadrature.

3.4 Mixed Form Generation

3.4.1 The `TimeFormGenerator` Class

The `TimeFormGenerator` class is a base class for all generators that take a weak space-time variational problem specified in UFL and its Irksome time-dependent extensions. Its goal is to serve as a template for subclass generators that create time-independent forms representing the variational problems on a variety of time-domain finite elements.

Before understanding the interface that a time form generator defines, it is important to present how the space-time block-matrix (in the case of linear problems) and block-vector (for nonlinear problems) systems can be represented by Firedrake `MixedFunctionSpace`s. In the following, "representative function space" refers to the Firedrake `FunctionSpace` used in the UFL with Irksome representation of space-time weak variational problems whereas "effective function space" refers to the `MixedFunctionSpace` that is actually employed by the Firedrake solver to solve the spacetime problem. It is to be noted that, in principle, the representative function space can be a mixed space as well, although in the current implementation of Fetsome mixed space formulations are not yet supported (see 5.2). It can therefore be said that a form generator maps a space-time weak variational problem on a representative function space to a spatial weak variational problem on an effective function space so that it can be solved. This correspondence is underpinned by the tensor product nature of the space-time function space.

Consider the linear unforced heat equation in weak variational form on a single degree p time continuous Petrov-Galerkin element. Then by (2.2.24) we have a linear system of equations to be solved encoded by the following block matrix-vector system:

$$\begin{bmatrix} I & 0 & \cdots & 0 \\ G^{(0,0)} & G^{(1,0)} & \cdots & G^{(p,0)} \\ \vdots & \vdots & \ddots & \vdots \\ G^{(0,p-1)} & G^{(1,p-1)} & \cdots & G^{(p,p-1)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_p \end{bmatrix} = \begin{bmatrix} \rho \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.4.1}$$

With $G^{(n,s)} = P^L_{ns} M + \Delta t P^M_{ns} K$, P^L being the time half-stiffness on trial function matrix and P^M the time mass matrix, M and K the standard spatial mass and stiffness matrices.

Now define a Firedrake $p+1$-dimensional mixed function space `Vt` from the exclusively spatial representative function space `V` of the weak variational problem (specified in UFL). Let, `uhat` and `vhat` be trial and test mixed-space functions for `Vt`. Then, sub-component `i` of `uhat` and `vhat` represents the restriction of finite element functions $u, v \in W_{kh}$ at the single timestep node `i` within a single time element. In the following code, subcomponents are accessed by `split`ting on each function.

```
V = FunctionSpace(mesh, ..., ...) # Dots for arbitrary spatial elements
Vt = MixedFunctionSpace(V, V, ..., V) # V repeated p+1 times
```

```
3
4  uhat = TrialFunction(Vt)
5  vhat = TestFunction(Vt)
6
7  uhatjs = split(uhat)
8  vhatis = split(vhat)
```

In the above, `uhatjs` and `vhatis` then are arrays containing each indexed spatial-only function being the restriction of the function at a single time-node. Assume that the Petrov-Galerkin time half-stiffness P^L is encoded by the matrix `P_L[n][s]` and P^M by `P_M[n][s]`. Then by the correspondence of each indexed sub-function to a timepoint of the time element, the single row `i` of the block system is equivalent to:

```
(self.P_L[0,i] * uhatjs[0] * vhatis[i]
    + self.dt * self.P_M[0,i] * dot(grad(uhatjs[0]), grad(vhatis[i])) + ...
    + self.P_L[p,i] * uhatjs[p] * vhatis[i]
    + self.dt * self.P_M[p,i] * dot(grad(uhatjs[p]), grad(vhatis[i])))*dx
    = Constant(0)*vhatis[i]*dx
```

It is important to note that each double term in the sum is a replacement `[v` $\mapsto$ `vhatis[i], u` $\mapsto$ `uhatjs[j]]` on the original form, an operation that can easily be handled by UFL's `replace` utility discussed in 3.2. By the Cartesian product nature of mixed function spaces (hence the orthogonality of all `fhatis[i]` sub functions for a mixed-space function `fhat`), all lines of the block system translated into Firedrake for a single `vhatis[i]` can be added together into a single UFL form without reducing the dimensionality of the system to be solved. For an arbitrary block Figure 3.5 schematically presents how an arbitrary block system's rows and columns corresponds to the two UFL form that Firedrake can solve for.

A `TimeFormGenerator` defines the interface for this correspondence programmatically. It receives a mapping of functions in the representative space (used in the user-supplied UFL weak form) to functions in the effective space to provide the foundation for all block-system replacement-based form generators.

3.4.2 The Splitting Form Generation Algorithms

Despite the block system to UFL form correspondence illustrated in the previous section and figure 3.5, the form composition strategy only based on UFL `replace`ment in the bilinear system is not general enough. In fact, it relies on using the effective function space to represent the matrix-vector block system as a single expression, but nonlinear systems do not reduce to such block systems (from section 2.2.7). Moreover, there is no other special case to which nonlinear systems reduce. To supply the generality needed, Fetsome introduces a new and original layer of form manipulation based on splitting forms on the order of time derivatives on test and trial functions, expansion in their time bases, and finally UFL replacement.

The algorithms and structures used within this layer are generalised for time cPG and dG finite elements, so that all core functionality for this kind of form generation is shared inside the `SplitTimeFormGenerator` subclass of `TimeFormGenerator`. The forms that a `SplitTimeFormGenerator` creates cannot directly be used to solve a FET problem, as they

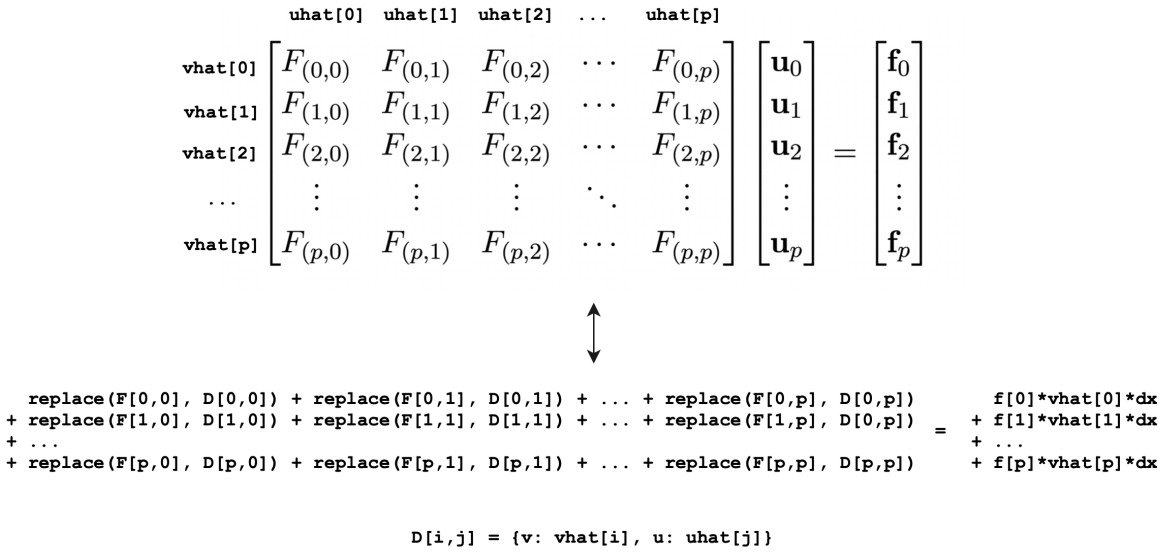


Figure 3.5: The correspondence between an arbitrary block system to UFL left-hand-side and right-hand-side forms by replacement

need to be supported by strongly or weakly imposed time boundary conditions that propagate information across time elements. Therefore, these forms are referred to as interior forms, as they only belong to the element considered independently.

When a weak variational form is passed through the `SplitTimeFormGenerator`, it is guaranteed to be linear in the test function v. Moreover, the implementation assumes that all time derivatives on the trial functions are expanded without loss of generality:

$$\frac{\partial}{\partial t} f(u) = f'(u) \frac{\partial u}{\partial t} \tag{3.4.2}$$

One further restriction is imposed on the form passed to the generator: explicit time dependence from the `t` variable should only come from forcing functions and boundary terms, that is explicit time dependence is additive and not multiplicative. This is discussed in more detail in 5.2. Therefore the general structure of forms integrated over a time interval that can be supplied to the `SplitTimeFormGenerator` is the following:

$$F_s = \int_{I_n} \int_{\langle \Omega, \partial\Omega, \Gamma \rangle} \left[g_0(u(x,t),x) \cdot h_0(v) + g_1(u,x) \cdot h_1\left(\frac{\partial v}{\partial t}\right) + ... + g_n(u,x) \cdot h_n\left(\frac{\partial^n v}{\partial t^n}\right) \right] d\langle x, s, S \rangle dt \tag{3.4.3}$$

Where $g_0, ..., g_n$ can be nonlinear in their arguments and can include arbitrary time and space derivatives of u, and $h_0, ..., h_n$ must be linear by construction but can include arbitrary space derivatives of v. The angled brackets represent a choice of integration domain and measure. Linear combinations $c_0 F_s^{(0)} + ... + c_m F_s^{(m)}$ of these forms are also supported by the generator. In Firedrake and Irksome, the forms supplied to the `SplitTimeFormGenerator` are `ufl.form.Form`s.

Figure 3.6 illustrates all the steps of the split and replace procedure illustrated in this section. The first step is to separate the supplied form in the orders of the time derivative applied to

v. This is always possible by the linearity of the form with respect to the test function. This yields:

$$[H_0(u,x;v), H_1(u,x;v), \cdots, H_n(u,x;v)] \tag{3.4.4}$$

$$\text{with} \quad H_i(u,x;v) = \int_{I_n} \int_{\langle \Omega, \partial\Omega, \Gamma \rangle} g_i(u(x,t),x) \cdot h_i\Big(\frac{\partial^i v}{\partial t^i}\Big) d\langle x,s,S\rangle dt \tag{3.4.5}$$

$$= \int_0^1 \int_{\langle \Omega, \partial\Omega, \Gamma \rangle} g_i(\tilde{u}(x,\tau),x) \cdot \Big(\frac{1}{\Delta t^{i-1}}\Big) h_i\Big(\frac{\partial^i \tilde{v}}{\partial \tau^i}\Big) d\langle x,s,S\rangle d\tau \tag{3.4.6}$$

In Fetsome, splitting is carried out using a recursive UFL expression tree traversal algorithm, implemented by `split_time_form_on(F, v)`. Within it, expression nodes have a default order of -1, each `v` expression node acquires an order of 0, instances of `Dt` on any expression except `v` are ignored and the order of all forms increases when an instance of `Dt` is applied to `v`. For efficiency in form reconstruction when splitting, the implementation exploits GEM's and TSFC's memoization functionality through its `Memoizer` and `MemoizerArg` classes used with `ufl_reuse_if_untouched`.

Once the original form is split on the orders of the test function, the next step is the expansion of the trial function from the representative to the effective mixed function space. This is done in a generic fashion to the Burgers' equation example in 2.2.7. Since UFL does not support a generic time variable, this substitution is performed for each quadrature point used to evaluate the time integral. It uses Fetsome's `spacetime_dot` to evaluate the linear combination of spatial basis functions at a time point equivalent to `u` or any of its time derivatives, followed by UFL `replace` to perform substitution. The resulting object is now a two dimensional collection of forms partially integrated in time, as shown in blue/first dashed line in figure 3.6.

Subsequently, all remaining Irksome `Dt` expressions nodes appear only applied to the test functions in the UFL expression tree. They can therefore be removed by the `strip_dt_form(F)` utility without losing knowledge on their order due to the previous splitting. In fact, `Dt` nodes cannot be understood by Firedrake's standard solution layer as they are only part of its Irksome extension and must be removed. Once again, this UFL expression tree traversal algorithm uses the same GEM and TSFC memoization utilities.

The last step is an expansion, like the one for the trial functions, of the test functions into the effective space using their time finite element space. After this, the indices of the remaining object are contracted to produce a single form, representing the spacetime linear or nonlinear system in the interior of the element to be solved. It is worth adding that, for linear problems, this split and replace procedure produces the same form as the block matrix system of figure 3.5. The obtained form can now be augmented with initial values and forcing terms.

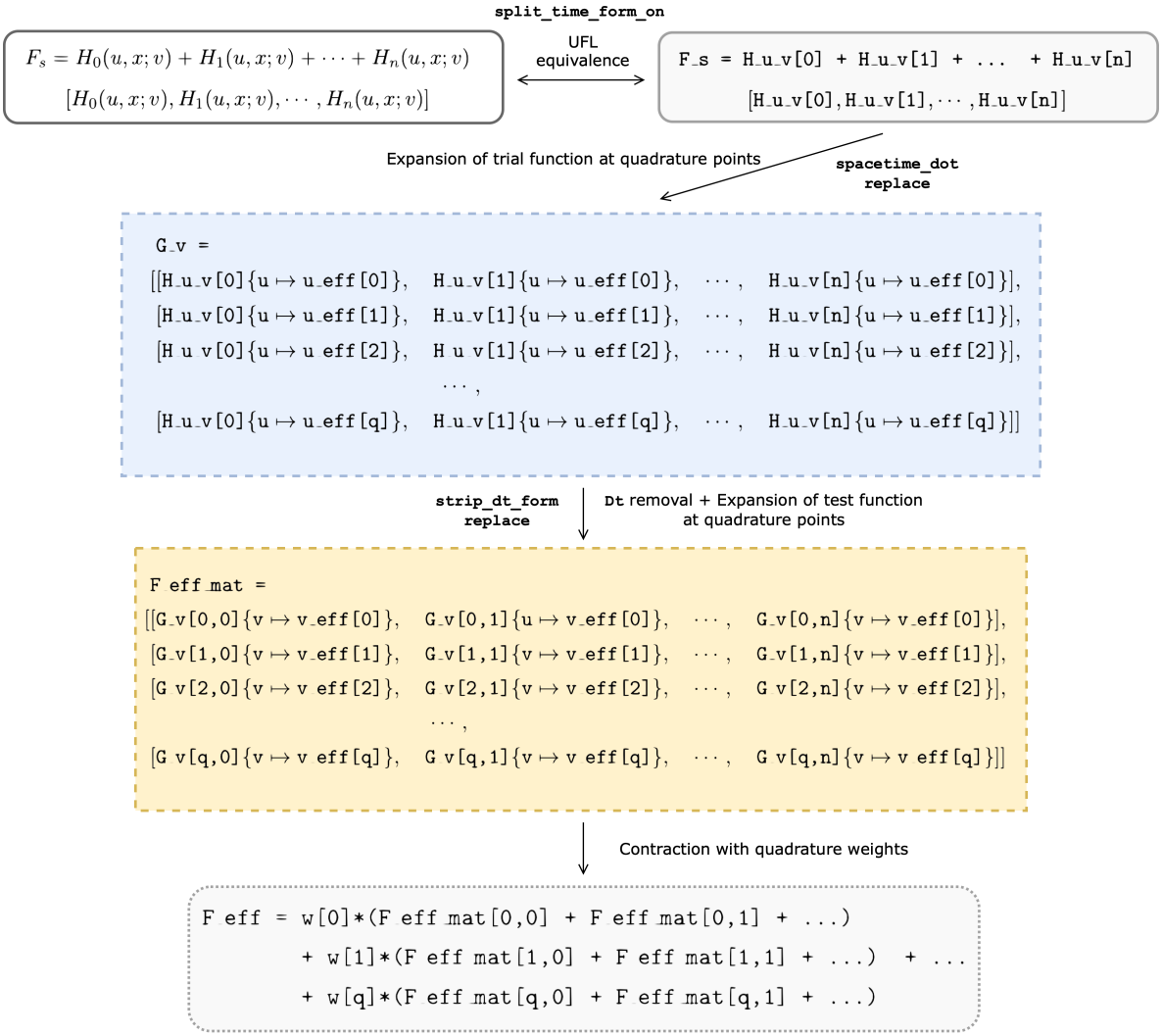


Figure 3.6: Representation of the split and replace procedure used by the `SplitTimeFormGenerator` to create interior forms

3.4.3 Adding Forcing Forms

Fetsome supports forcing forms in the specification of weak variational problems. Consider the following integral over a timestep, where the forcing function $f(x,t)$ is known:

$$L(v) = \int_{I_n}\int_{\Omega} f(x,t)v(x,t)dxdt = \Delta t \int_0^1 \int_{\Omega} \tilde{f}(x,\tau)\tilde{v}(x,\tau)dxd\tau \tag{3.4.7}$$

Then by expansion of v and f into their finite element bases and introduction of time quadrature (using Einstein summation convention):

$$\begin{aligned} L(v) =& \Delta t \int_0^1 \int_{\Omega} f_{ij}\tilde{\psi}_i(\tau)\phi_j(x)v_{kl}\tilde{\psi}'_k(\tau)\phi_l(x)dxd\tau && (3.4.8)\\ \approx& \Delta t f_{ij}v_{kl}b_m\tilde{\psi}_i|_{\tau_m}\tilde{\psi}'_k|_{\tau_m}\int_{\Omega}\phi_j\phi_l dx && (3.4.9)\\ =& \Delta t f_{ij}v_{kl}Q^M_{ik}M_{jl} && (3.4.10) \end{aligned}$$

Q_{ik} is a quadrature-dependent time mass matrix (either P_{ik} for cPG or D_{ik} for dG). Picking $v_{kl} = \delta_{ks}\delta_{lr}$:

$$\begin{aligned} &= \Delta t f_{ij}\delta_{ks}\delta_{lr}Q^M_{ik}M_{jl} = \Delta t f_{ij}Q^M_{is}M_{jr} \\ &= \Delta t(f_{0j}Q^M_{0s}M_{jr} + ... + f_{pj}Q^M_{js}M_{jr}) \end{aligned} \tag{3.4.11}$$

Therefore, if `fs` is the vector containing the Firedrake `Function`s of f evaluated at the time points of the time element, the UFL equivalent for the full system is:

```
F_f = Constant(dt)*(Q_M[0,0]*fs[0] + ... + Q_M[p,0]*fs[p])*vhatis[0]
      + Constant(dt)*(Q_M[0,1]*fs[0] + ... + Q_M[p,1]*fs[p])*vhatis[1]
      + ...
      + Constant(dt)*(Q_M[0,p]*fs[0] + ... + Q_M[p,p]*fs[1])*vhatis[p]
```

3.4.4 Adding Boundary Terms

One of the goals of Fetsome is to support the integration of space-time weak boundary terms with standard Firedrake. This is fundamental for the movement of partial space-time derivatives from trial functions to test functions for non-smooth solutions, and necessary for the weak enforcement of initial conditions for time-discontinuous elements. The layer of Fetsome handling boundary terms is also part of the `SplitTimeFormGenerator` as it uses a similar substitution strategy. To understand how this layer works, the two kinds of boundary integrals to consider are space domain boundary forms and time interval boundary forms.

Firedrake, through UFL, divides integrals over interior and exterior facets of an exclusively spatial domain by the integration measure used. Like `dx` is a measure for spatial integration, similarly `dS` is the measure for integration over domain interior facets and `ds` integrates domain exterior facets. To understand how these integrals interact with the time discretisation, the following commonly-found example is considered:

$$\int_0^1 \int_{\partial\Omega} v\nabla u \cdot \mathbf{n} dS d\tau \tag{3.4.12}$$

$$= \int_0^1 \int_{\partial\Omega} v_{kl}\psi'_k(\tau)\phi_l(\mathbf{x})u_{ij}\psi_i(\tau)\nabla\phi_j(\mathbf{x}) \cdot \mathbf{n} dS d\tau \tag{3.4.13}$$

$$= u_{ij}v_{kl}\left(\int_0^1 \psi_i(\tau)\psi'_k(\tau)d\tau\right)\left(\int_{\partial\Omega} \phi_l(\mathbf{x})\nabla\phi_j(\mathbf{x}) \cdot \mathbf{n} dS\right) \tag{3.4.14}$$

$$= u_{ij}v_{kl}Q^M_{ik}\left(\int_{\partial\Omega} \phi_l(\mathbf{x})\nabla\phi_j(\mathbf{x}) \cdot \mathbf{n} dS\right) \tag{3.4.15}$$

Note that the Einstein summation convention is used. Q_{ik} is a quadrature-dependent time mass matrix. This shows that for the shown space-time implementation with splitting, space boundary terms will pass-through and be handled by standard Firedrake, therefore these terms are supported without any modification to the implementation.

On the other hand, Fetsome must treat time boundary forms in a special way. For this,

the following example is considered:

$$\left[\int_\Omega \tilde{u}\tilde{v}dx\right]_0^1 = \left[\int_\Omega \tilde{u}\tilde{v}dx\right]_{\partial I} \tag{3.4.16}$$

In which for a specific point in time τ^*:

$$\int_\Omega \tilde{u}\tilde{v}dx = \int_\Omega u_{ij}\psi_i(\tau^*)\phi_j(x)v_{kl}\psi'_k(\tau^*)\phi_l(x)dx \tag{3.4.17}$$

$$= u_{ij}\psi_i(\tau^*)v_{kl}\psi'_k(\tau^*)\int_\Omega \phi_j(x)\phi_l(x)dx = u_{ij}\psi_i(\tau^*)v_{kl}\psi'_k(\tau^*)M_{jl} \tag{3.4.18}$$

$$\implies \left[\int_\Omega \tilde{u}\tilde{v}dx\right]_{\partial I} = u_{ij}\psi_i(1)v_{kl}\psi'_k(1)M_{jl} - u_{ij}\psi_i(0)v_{kl}\psi'_k(0)M_{jl} \tag{3.4.19}$$

For a time-step set of nodes that includes the endpoint, let $\perp$ represent the index of the time nodal basis function which is 1 at $\tau = 0$, $\top$ the index for the basis function which is 1 at $\tau = 1$. Therefore for arbitrary test function coefficients v_{kl}:

$$\left[\int_\Omega \tilde{u}\tilde{v}dx\right]_{\partial I} = \left[u_{ij}\psi_i(\tau^*)v_{kl}\psi'_k(\tau^*)\right]_0^1 M_{jl} = (u_{\top j}v_{kl}\delta_{\top k} - u_{\perp j}v_{kl}\delta_{\perp k})M_{jl} \tag{3.4.20}$$

We can then pick $v_{kl} = \delta_{ks}\delta_{lr}$, from arbitrariness of the test function with s a ranging temporal index and r a ranging spatial index. Thus follows:

$$\implies \forall s, r : (u_{\top j}\delta_{ks}\delta_{lr}\delta_{\top k} - u_{\perp j}\delta_{ks}\delta_{lr}\delta_{\perp k})M_{jl} = (u_{\top j}\delta_{\top s} - u_{\perp j}\delta_{\perp s})M_{jr} \tag{3.4.21}$$

$$= \begin{cases} u_{\top j}M_{jr} & s = \top \\ -u_{\perp j}M_{jr} & s = \perp \\ 0 & \text{otherwise} \end{cases} \tag{3.4.22}$$

Therefore, Fetsome can compute the contributions of the boundary terms by using the effective mixed space functions representing the solution at each time step's endpoints and only adding them when multiplied by the test function for the same endpoint. For a supplied UFL form `db` and test function `v` to be integrated over the reference time boundary, `SplitTimeFormGenerator._expand_boundary_term` performs splitting on the test function orders with `split_time_form_on(db, v)`. This is followed by the `replace`ment of each representative trial function `u` (and its time derivatives) with the first and last of the effective functions of `uhat` (and their time derivatives) on the time boundary or their weakly-enforced value on the time-boundary (as in section 3.6.1 for dG). Each boundary term is added only for the sub-functions of `vhat` that represent the endpoint at which it was evaluated.

3.4.5 Time Dependent Dirichlet Conditions

Dirichlet conditions in Firedrake are handled by the `DirichletBC` class, they are directly passed to the Firedrake solver. Enforcing time-dependent Dirichlet boundary conditions on the spatial domain Ω is then only a matter of replicating each `DirichletBC` object for a problem for each time point corresponding to the chosen time finite element basis functions. When each `DirichletBC` is enforced on the subspace of the effective function space directly corresponding to a time point in a timestep, the Firedrake solver will independently ensure that the

mixed space solution will satisfy each condition if the problem is well posed. The remaining, non-nodal part of the timestep then will satisfy the time-interpolated spatial Dirichlet condition, as required. If the boundary condition has no time dependence, then it still requires replicating for each of the time points.

Due to Firedrake interpolating a Dirichlet boundary condition on a space domain upon initialisation, Fetsome requires the `TimeDirichletBC` subclass to be used by its users whenever the condition is time-dependent (see section 3.5).

3.5 Interface to the Solver

As discussed previously, a space-time weak variational problem is defined using the Irksome `Dt` operator and explicit time dependence is given using the `t` variable declared as a UFL `Constant`. In order to define the temporal component of the tensor product space-time finite element space, the `TimeFunctionSpace` utility is used, which automatically provides the correct Firedrake `FunctionSpace` for the polynomial degree of the time domain requested. If time dependent Dirichlet boundary conditions are used in the problem, they must be specified as `TimeDirichletBC` objects, as to give the Fetsome expression manipulation layer an expression for each condition to be evaluated at element time points before Firedrake fixes the value of the constant `t` at interpolation. Once the space-time variational problem is formulated, all of its components are passed to the `VariationalTimeStepper`.

3.5.1 The `VariationalTimeStepper`

The Fetsome `VariationalTimeStepper` class aims to provide a similar interface to the Irksome `TimeStepper`s, although it fundamentally differs in its solution strategy. When given a full formulation of a space-time finite element problem, the time stepper class provides an `advance` method which solves the problem over a single timestep and produces the Firedrake `Function`s representing the solution across the timestep. Subsequent calls to `advance` continue the stepping procedure over following elements. As FET produces intermediate interior function values across a single time step which are meaningful, differently from most Runge-Kutta schemes in which the intermediate values are just stages used to compute a weighted average of gradients, `advance` gives the user the choice to obtain only the final solution or an array containing the function restricted to each of a time-step's points.

The components which a `VariationalTimeStepper` instance requires for a complete FET problem specification are:

- A triple `F = (b, db, L)` where `b` is the sum of all forms which contain integrals over the full time domain, `db` is the sum of the forms containing integrals over the timestep boundary, `L` containing all linear forcing forms
- An initial condition `u0`, the trial function `u` and test function `v` as Firedrake `Function`s from the problem's representative function space
- The function space `T` representing the time domain of the space-time tensor product space

- The `t` time variable and the timestep size `dt`
- The choice of time finite element between cPG and dG
- Optionally, it can take space domain boundary conditions, PETSc solver parameters to pass to the solver and a manually selected `TimeQuadrature` scheme

Through the choice of time finite element family, the class creates the effective trial and test function spaces needed for the problem, composes the representative to effective space mappings and initialises the corresponding `TimeFormGenerator`. By collecting the initial condition and sampling the space-time forcing terms at the time points of the chosen time finite element, the `VariationalTimeStepper` invokes its `TimeFormGenerator` to compose the mixed forms that can be passed to the the standard Firedrake solver. If spatial Dirichlet boundary conditions are present, they are converted to effective space spatial-only conditions as previously outlined. The time step forms and boundary conditions are then passed to a call to Firedrake's `solve`. Section 3.6.2 discusses the additional details of the `VariationalTimeStepper` that allow it to solve nonlinear problems too.

3.5.2 Solving With Petrov-Galerkin Elements

When passing `family="CPG"` to the `VariationalTimeStepper`, it initialises the test and trial function spaces in time to differ by one in their polynomial orders, as required in section 2.2.3, and selects a `PetrovTimeFormGenerator` to produce the time step forms.

The `PetrovTimeFormGenerator` imposes the initial condition of the timestep strongly by using the first test function of the effective function space (corresponding to the first line of the block system in 3.4.1 which is retained in both the linear and nonlinear cases). To produce the interior forms, it supplies the remaining effective space test functions to the shared form generation algorithms implemented by its `SplitTimeFormGenerator` parent class, which composes them using the split and replace procedure. The `PetrovTimeFormGenerator` also adds the contributions from forcing and time boundary terms, which use the effective space test functions instead of weakly imposed initial or final values, after which it returns the full UFL form representing the system to the `VariationalTimeStepper` for solution.

3.5.3 L^2 Spacetime Error

In addition to the functionality presented, Fetsome provides a utility function to compute the space-time L^2 error of equation (2.2.7) between an exact solution and a vector of Firedrake functions corresponding to the time step values of the finite element solution: `time_errornorm`. Since the space-time function space is composed as a tensor product of temporal and spatial bases rather than their Cartesian product, the error cannot be computed pointwise at the time points of each element and added. Instead, it involves quadrature of the time integral in (2.2.7) and the expansion of the numerical solution at the quadrature points through the use of `spacetime_dot`.

3.6 Advanced Implementation Details

The implementation presented thus far is designed to be flexible for extensions in form generation and in the type of problems that it supports. Nevertheless, there are additional

implementation details to be considered for discontinuous Galerkin elements and the support for nonlinear problems.

3.6.1 Solving With Discontinuous Galerkin Elements

A discontinuous Galerkin time discretisation can be selected by passing the `family="DG"` to a `VariationalTimeStepper` upon initialisation. This sets the trial and test spaces in the time domain to be equal to one another, then supplies additional forcing functions and boundary terms to the `DiscontinuousTimeFormGenerator` to produce the UFL forms passed to the Firedrake solver.

A `DiscontinuousTimeFormGenerator` does not impose any initial conditions strongly, as this would guarantee continuity between neighbouring intervals and reduce the degrees of freedom available for a discontinuous discretisation. Initial conditions are imposed weakly through the time-boundary conditions. As detailed in 2.2.4 the time dG discretisation implemented by Fetsome uses time upwind flux. Therefore, between elements, the value of the sought trial function is always chosen to be that of the element belonging to the previous interval. Within the implementation of `_expand_boundary_term`, used by the `SplitTimeFormGenerator`, it is possible to pass a known function to be used instead of the trial function `u` at the lowest or highest time point of time-boundary terms. In the case of upwind flux, the initial condition is used at the lowest time point. When the time element is not the first in the time stepping routine, the `VariationalTimeStepper` supplies the value of the solution at the final node of the previous element as the initial condition to the `DiscontinuousTimeFormGenerator`. The highest time point boundary value is left unspecified, as the default behaviour of the generator is to use the trial function for the last node of the time step, as needed by upwind flux.

3.6.2 Solving Time Dependent Nonlinear Problems

The central functionality enabling the solution of nonlinear problems in Fetsome is the split and replace procedure of generating space-time-equivalent UFL forms. No direct distinction is made by the Fetsome layer between linear and nonlinear problems, so each weak variational problem is manipulated in the generality required by a nonlinear problem.

The first important detail in preparing a Firedrake nonlinear problem is to obtain a residual form (introduced in section 2.2.7) that vanishes for all solutions to the problem. This is done by the `VariationalTimeStepper` for the space-time problem, once a `PetrovTimeFormGenerator` or a `DiscontinuousTimeFormGenerator` has generated the effective space forms that solve the FET problem on the considered time step.

The second important detail regards the convergence of the iterative scheme used to solve the nonlinear problem. Without an initial guess close to the true solution of the scheme, it might not be possible for convergence to this true solution to be achieved. The heuristic used in Fetsome is to supply the initial value (or the last solution value of the previous timestep) as a guess for the solution across the entire time step. As the chosen time step size is decreased, it is expected that this heuristic is increasingly correct.

Chapter 4

Evaluation

This chapter presents an evaluation of the mathematical correctness of the finite element in time software formulation as implemented in Fetsome. As has been shown in Chapter 3, FET problems present a diverse set of mathematical objects that can be used in their variational formulation. The most fundamental, which require thorough evaluation to analyse the success of the presented implementation, are the following:

- Bilinear or nonlinear forms with time derivatives applied to trial functions only
- Bilinear or nonlinear forms with time derivatives applied to test functions
- Linear forms produced by integrating the product of test functions and forcing functions
- Time boundary terms that appear when shifting time derivatives onto test functions
- Space boundary terms that appear when shifting spatial derivatives onto test functions
- Strong and weak initial conditions for the different cPG and dG discretisations
- Spatial operators that are defined based on the spatial dimension of the problem (e.g. in 1 dimension $\nabla \cdot u = \frac{\partial u}{\partial x}$, in 3 dimensions $\nabla \cdot u = \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z}$)

By considering a variety of important model problems, it is highlighted how the presented implementation successfully tackles each of these. For this task, convergence testing allows for a strictly quantitative evaluation procedure, as described in detail in section 4.1. Tests are performed in different spatial dimensions, to guarantee the implementation's flexibility and solidity in seamlessly handling FET problems arbitrary of their spatial discretisation.

Given this investigation's goal to extend Firedrake and Irksome's capabilities, this evaluation section also briefly highlights the design choices that integrate the presented extensions into a typical user's experience with time-dependent problems. All the problems considered as part of this section are available on the Fetsome branch in Irksome along with their convergence tests (https://github.com/firedrakeproject/Irksome/tree/jack-lascala/fetsome).

4.1 Convergence Testing Methodology

Convergence testing is the principal code verification method for the success of the presented software. This consists in examining the scaling of the L^2 error computed in spacetime, as

the time mesh parameter (the timestep Δt) varies, between the numerical solution obtained by the presented solver and an expected solution. Despite the implementation exploiting Firedrake mixed spaces to obtain spacetime solutions, it is to be noted that any spacetime norm is not simply computed as the sum of space and time errors as in mixed spaces. This is discussed in section 3.5.3.

Consider a FET discretisation that uses degree k polynomial 1D finite elements for the time-domain with typical mesh size Δt (the timestep) and a spatial discretisation with typical mesh size h that converges with error $\mathcal{O}(h^m)$ for some exponent m. As discussed in sections 2.2.3 and 2.2.4, we have an expected L^2 spacetime error that asymptotically scales as $\mathcal{O}(\Delta t^{k+1})$, assuming a theoretically perfect spatial discretisation. Nevertheless, the inevitable numerical approximation of the spatial domain effectively modifies the asymptotic scaling of the error to be $\mathcal{O}(h^m + \Delta t^{k+1})$. This implies that as one parameter between h and Δt varies, there will be an interval of transition between which of the two error terms asymptotically dominates. Hence, for a fixed h it is expected that as $\Delta t \to 0$ we will not have $||u_{kh} - u||_{L^2} \to 0$, rather the error will tend to some constant depending on h. To mitigate this error for evaluation purposes, the spatial discretisation parameter h is decreased for the smallest chosen Δt until the error numerically converges to a constant (the error for the specific value of Δt), then all subsequent time steps are analysed with such a fixed h. It is worth noting that the degree of spatial elements in the case of polynomial spatial bases also affects the convergence parameter m, although its modification is discouraged from its costly impact on the time taken to obtain solutions.

Due to the inevitable computational limitations in floating point accuracy, a break down of the error's asymptotic behaviour is expected as $h, \Delta t \to 0$. On the other hand, the asymptotic behaviour is defined for both parameters tending to zero, therefore an upper bound on the range of Δt where the error scales correctly is also expected. Each of the examples discussed in this section present and argue such an interval of validity for the convergence rates obtained.

4.2 Method of Manufactured Solutions

Partial differential equations of interest rarely have solutions that can be obtained in closed form. It is therefore important to use the method of manufactured solutions as a way to obtain the exact solutions needed to compute L^2 errors to evaluate numerical solutions. Consider the following PDE, to which boundary conditions of any kind are associated:

$$F(u, x, t) = f(x, t) \tag{4.2.1}$$

In the above, F can include an arbitrary number of space and time derivatives, as well as arbitrary nonlinearities. Instead, $f(x, t)$ will be called a forcing function and cannot include u or any of its derivatives. It follows that, if $f(x, t)$ is left unspecified, any determined function $\hat{u}(x, t)$ that satisfies the given boundary conditions solves the following forced PDE:

$$F(u, x, t) = F(\hat{u}(x, t), x, t) \tag{4.2.2}$$

Hence, a forced alternative to a PDE of interest can always be obtained by first choosing $\hat{u}$ and subsequently taking $f(x,t) = F(\hat{u}(x,t), x, t)$. This way of constructing PDEs with known closed form solutions is the method of manufactured solutions [33].

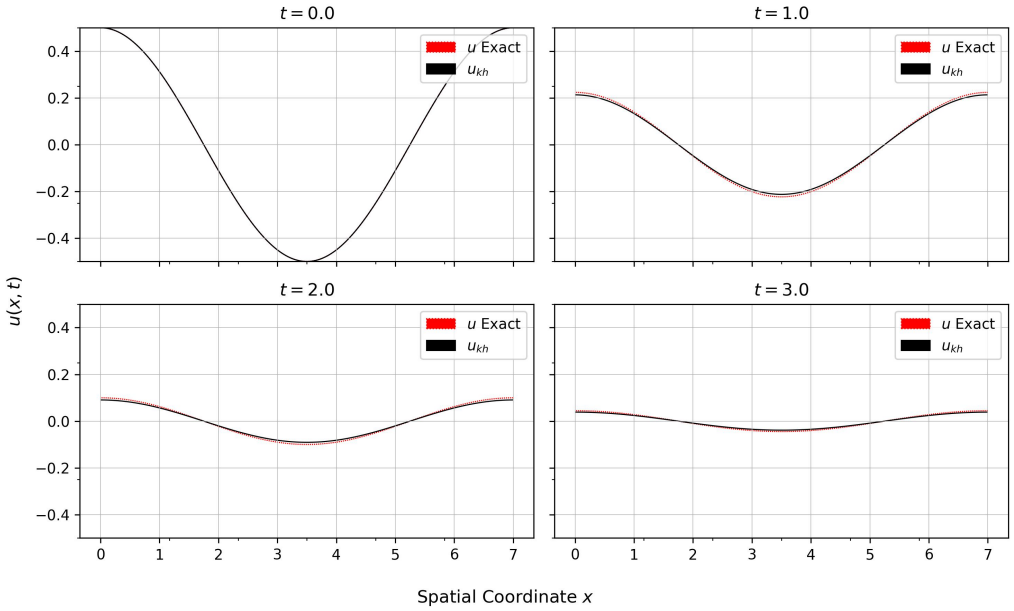


Figure 4.1: Snapshots of the evolution of the 1D homogenous heat equation on $0 \leq t \leq 3$ compared to the analytic solution; linear cPG time finite elements, $\Delta t = 1.0$

4.3 Analytic 1D Heat Equation

The first example that is considered to evaluate the most fundamental functionality of the presented implementation, namely the solution of unforced equations without boundary terms or Dirichlet conditions, is the heat equation. The homogenous heat equation with Neumann boundary condition reads as follows: on a spatial domain $x \in \Omega$, time domain $t \in [0, T]$, for a function $u_0(x)$, find $u(x,t)$ such that:

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0 \tag{4.3.1}$$

$$u(x,0) = u_0(x) \tag{4.3.2}$$

$$\nabla u \cdot \mathbf{n} = 0 \text{ on } \partial\Omega \tag{4.3.3}$$

When restricted to 1 spatial dimension, we have $\nabla^2 = \frac{\partial^2}{\partial x^2}$ and $\Omega = [0, L]$. The corresponding variational form of the problem, using continuous Petrov-Galerkin elements with trial and test spaces W_{kh} and W'_{kh}, is to find $u_{kh} \in W_{kh}$ such that $\forall v_{kh} \in W'_{kh}$ the following holds:

$$\int_0^T \int_0^L \frac{\partial u_{kh}}{\partial t} v_{kh} dx dt + \int_0^T \int_0^L \frac{\partial u_{kh}}{\partial x} \frac{\partial v_{kh}}{\partial x} dx dt = 0 \tag{4.3.4}$$

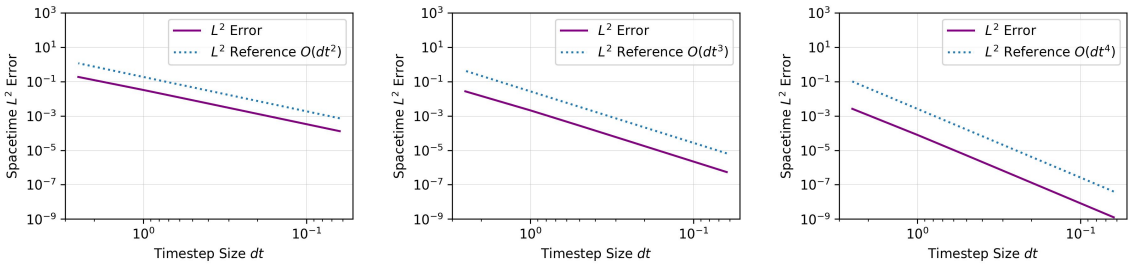
If the time and space domains are picked with $T = 5$, $L = 7$ along with the initial condition $u_0(x) = \frac{1}{2}\cos(\frac{2\pi x}{7})$, it can be verified that the specified problem has analytic closed form

solution $u(x,t) = \frac{1}{2}\cos(\frac{2\pi x}{7})\exp(-\frac{4\pi^2 t}{49})$. For the Fetsome specification, continuous Petrov-Galerkin elements are picked to discretise the time mesh coupled to continuous Lagrange spatial elements. The `VariationalTimeStepper` solver for the problem can then be set up and initialised as illustrated below.

```
mesh = IntervalMesh(Ns, 7)
V = FunctionSpace(mesh, "CG", 2)

time_interval = UnitIntervalMesh(1)
T = FunctionSpace(time_interval, "CG", kt)
u = TrialFunction(V)
v = TestFunction(V)
b = (Dt(u)*v + dot(grad(u), grad(v))) * dx

x = SpatialCoordinate(mesh)[0]
t = Constant(0)
u0 = interpolate((1/2)*cos((2*pi/7)*x), V)

F = (b, None, None)
timestepper = VariationalTimeStepper(F, u0, u, v, T, t, dt, "CPG")
```

In the initialisation code, `dt` holds the chosen timestep size, `kt` holds the degree for the time finite elements and `Ns` holds the number of spatial elements. Timestepping is then performed by calls to `timestepper.advance()` to obtain for $0 < t < 5$ a solution similar to the one presented in figure 4.1.

To evaluate the rate of convergence of the numerical solution to the exact solution, the timestep parameter Δt is varied between the values $0.0625, 0.125, 0.25, 0.5, 1.0, 2.5$. A choice of 400 quadratic spatial elements were found to make the space-time error converge within 0.1%. To demonstrate that the correct expected convergence rates are achieved for different time elements, linear to cubic elements are examined. The errors for each time step and polynomial degree are available in appendix A.1. Figure 4.2 illustrates the scaling of the L^2 spacetime error for the three chosen time finite elements. As seen, all elements converge with their expected rates: linear elements converge quadratically, quadratic elements con-



(a) Linear time elements (b) Quadratic time elements (c) Cubic time elements

Figure 4.2: L^2 errors for the 1D homogenous heat equation solution with correct convergence rates for linear, quadratic and cubic time finite elements

verge cubically and cubic elements converge quartically. It is clear from the figure that the expected asymptotic behaviour is obtained for all the timestep sizes of interest, therefore we can conclude that the numerical solution for this problem is in fact obtained correctly.

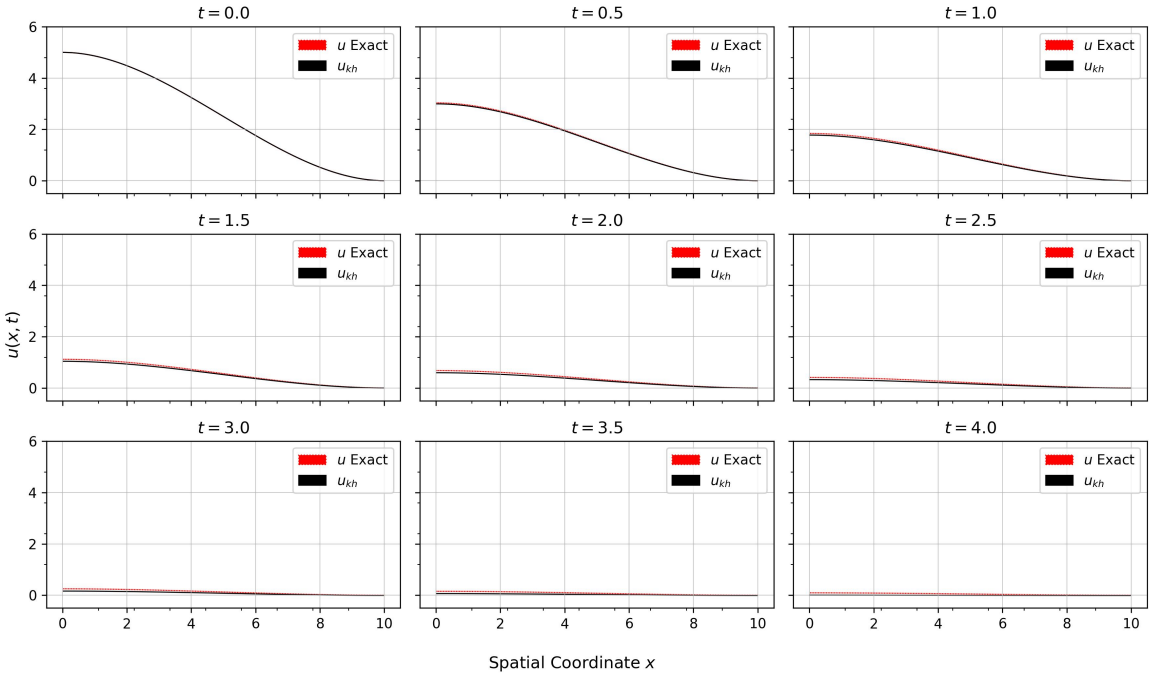


Figure 4.3: Snapshots of the evolution of the 1 dimensional heat equation subject to the forcing function $f(x,t) = \left(-1/100x^3 + 3/20x^2 - 3/50x - 53/10\right)\exp(-t)$; linear dG time elements, $\Delta t = 0.5$

4.4 1D Forced Heat Equation

Similar to the homogenous or unforced heat equation, we introduce a time-dependent forcing function $f(x,t)$ and modify equation (4.3.1) to the following:

$$\frac{\partial u}{\partial t} - \nabla^2 u = f(x,t) \tag{4.4.1}$$

Homogenous Neumann boundary conditions are maintained. The forced heat equation presents an opportunity for evaluating Fetsome's support for discontinuous Galerkin elements in time, as well as its manipulation of boundary terms and time dependent forcing functions. Taking the spatial domain to be in a single dimension and using the same space W_{kh} for both trial and test functions, as required by the time dG discretisation, this leads to modification of the weak variational problem (4.3.4) to finding $u_{kh} \in W_{kh}$ such that $\forall v_{kh} \in W_{kh}$:

$$\sum_n \int_0^L [u_{kh} v_{kh}]_{\partial I_n} dx - \int_0^T \int_0^L u_{kh} \frac{\partial v_{kh}}{\partial t} dx dt + \int_0^T \int_0^L \frac{\partial u_{kh}}{\partial x} \frac{\partial v_{kh}}{\partial x} dx dt = \int_0^T \int_0^L f v_{kh} dx dt \tag{4.4.2}$$

For the specific problem, the parameters are taken to be $L = 10$, $T = 4$. It can be verified that the following, obtained through the method of manufactured solutions of 4.2, constitute an admissible solution to the forced heat equation and satisfy the homogenous Neumann

boundary condition for all intermediate times:

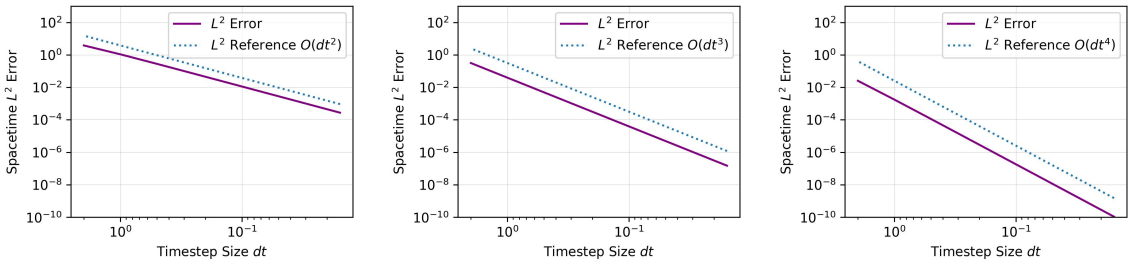
$$u_0(x) = \frac{1}{100}x^3 - \frac{3}{20}x^2 + 5, \quad f(x,t) = \left(-\frac{1}{100}x^3 + \frac{3}{20}x^2 - \frac{3}{50}x - \frac{47}{10}\right)e^{-t} \tag{4.4.3}$$

$$u(x,t) = \left(\frac{1}{100}x^3 - \frac{3}{20}x^2 + 5\right)e^{-t} \tag{4.4.4}$$

Choosing the `"DG"` family upon initialisation of the `VariationalTimeStepper`, the code presented in section 4.3 to solve the unforced 1D heat equation can be thus modified as follows to incorporate the forcing term and time-boundary term required by the weak variational problem.

```
... # Previous initialisation code
x = SpatialCoordinate(mesh)[0]
t = Constant(0)
u0 = interpolate(1/100 * x**3 - 3/20 * x**2 +5, V)
b = (-u*Dt(v) + dot(grad(u), grad(v))) * dx
db = u*v*dx
L = (-1/100 * x**3 + 3/20 * x**2 - 3/50 * x - 47/10) * exp(-t) * v * dx

F = (b, db, L)
timestepper = VariationalTimeStepper(F, u0, u, v, T, t, dt, "DG")
```

Figure 4.3 presents a comparison between the obtained FET solution and the exact solution when using linear time elements and a timestep size of 0.5. For the spatial discretisation, 400 quadratic elements were found to allow convergence of the error across all chosen timesteps (see appendix A.2). Figure 4.4 demonstrates the correct orders of convergence obtained by linear (quadratic convergence), quadratic (cubic convergence) and cubic (quartic convergence) elements. Once again, the validity of the asymptotic behaviour visibly extends to the whole range of time steps considered.



(a) Linear time elements (b) Quadratic time elements (c) Cubic time elements

Figure 4.4: L^2 errors for the time-dG 1D forced heat equation for varying timestep size with reference convergence rate lines

4.5 2D Linear Transport Equation on Periodic Domain

To evaluate the capability of the Fetsome layer to deal with multiple spatial boundary conditions in higher dimensions, the linear transport equation on a cylinder is considered. The

transport equation problem is to find $u(x, y, t)$ such that:

$$\frac{\partial u}{\partial t} + \mathbf{c} \cdot \nabla u = 0 \tag{4.5.1}$$

$$u(x, y, 0) = u_0(x, y) \tag{4.5.2}$$

In the above, $\mathbf{c}$ is the constant advection speed of the solution. Taking $\mathbf{c} = (c, 0)$, which represents advection in the x direction, the solution to the linear transport equation is:

$$u(x, y, t) = u_0(x - ct, y) \tag{4.5.3}$$

The FET weak variational form of the problem on the domain $\Omega \times [0, T]$ is to find $u_{kh} \in W_{kh}$ such that for all $v_{kh} \in W_{kh}$:

$$\int_\Omega [u_{kh} v_{kh}]_{\partial[0,T]} dx - \int_0^T \int_\Omega u_{kh} \frac{\partial v_{kh}}{\partial t} dx dt - \int_0^T \int_\Omega u_{kh} \mathbf{c} \cdot \nabla v dx dt = 0 \tag{4.5.4}$$

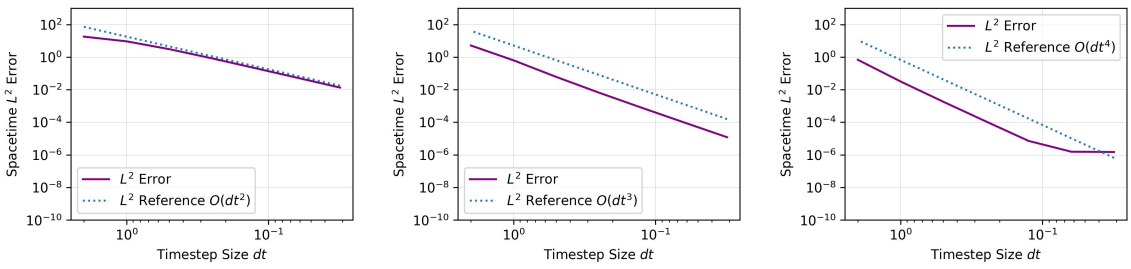
For the numerical solution using Fetsome, the domain Ω is chosen to be the rectangle $[0, 10] \times [0, 5]$ that is periodic in the x-direction, discretised through 150×75 continuous Lagrange spatial elements of degree 3. The advection speed is taken to be $\mathbf{c} = (5/2, 0)$ and the initial condition is taken to be:

$$u_0(x, y) = 5 \sin^2\left(\frac{\pi x}{10}\right) \sin\left(\frac{\pi y}{5}\right) \tag{4.5.5}$$

$$\implies u(x, y, t) = 5 \sin^2\left(\frac{\pi}{10}\left(x - \frac{5}{2}\right)t\right) \sin\left(\frac{\pi y}{5}\right) \tag{4.5.6}$$

The time domain is then picked to be $[0, T] = [0, 4]$, or one single period of the solution, discretised through continuous Petrov-Galerkin elements.

Figure 4.5 shows the obtained convergence orders for the transport equation problem (the data can be found in appendix A.3). Differently from the heat equation solutions, the ranges which present the sought asymptotic convergence rates are obtained is less clear. In fact, it can be seen that linear elements present a deviation from the convergence regime for time step sizes greater than $\Delta \sim \mathcal{O}(10^0)$. This can be ignored by noting that such an order of time step size is excessive for the chosen time domain. Similarly, it is evident that the $\Delta t \to 0$ limit for cubic elements breaks the $\mathcal{O}(\Delta t^4)$ expected convergence regime under $\Delta t \sim \mathcal{O}(10^{-1})$. This



(a) Linear time elements (b) Quadratic time elements (c) Cubic time elements

Figure 4.5: L^2 error of cPG 2D semi-periodic linear transport equation with reference convergence rates

results from the dominance of the spatial discretisation error, and can be considered expected and not influent on the correct convergence rate demonstrated in the rest of the considered range. For the remainder of the evaluated Δt ranges, all considered elements achieve their expected convergence rates. Therefore, we can conclude that the transport equation provides an example where the Fetsome functionality is successful in applying FET to more complicated and higher dimensional spatial domains.

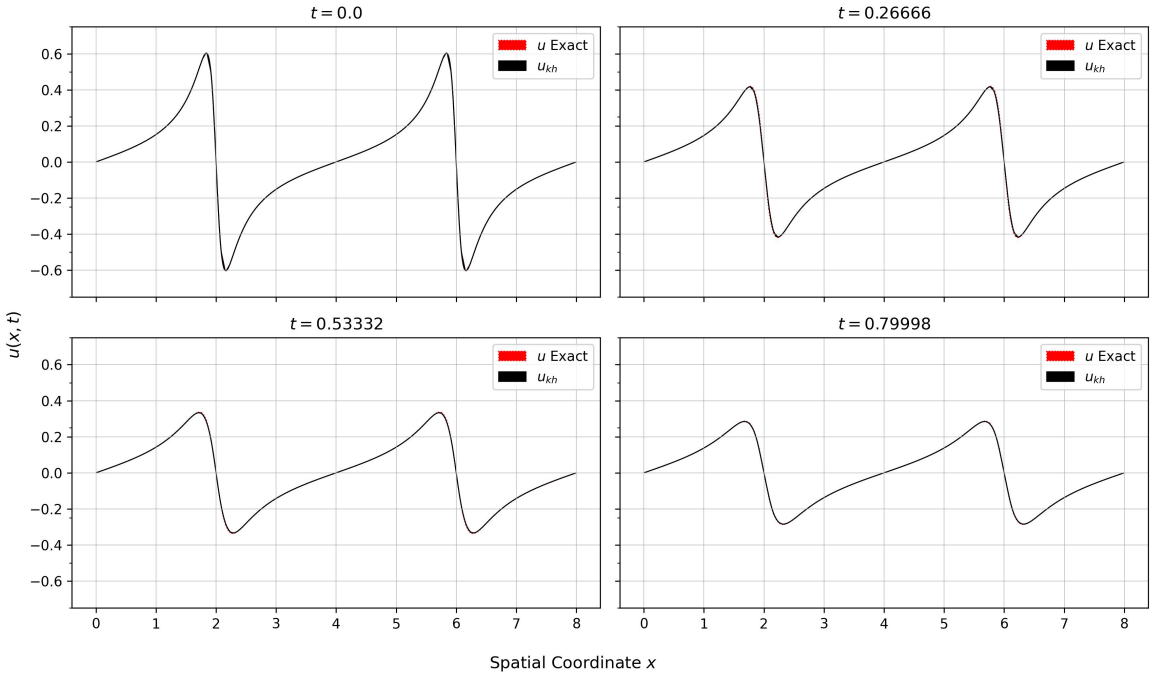


Figure 4.6: Snapshots of the exact and numerical solutions to the viscous Burgers' problem on $0 \leq t \leq 0.8$ (lines greatly overlapping); linear cPG elements, $\Delta t = 0.26666$

4.6 Continuous Viscous Burgers' Equation

In order to assess the convergence properties of the Fetsome implementation on nonlinear problems, the 1D viscous Burgers' equation is solved numerically. This problem is fundamental in fluid dynamics applications and in the study of advection diffusion equations, demonstrating the applicability of the FET implementation presented in this investigation. On a one dimensional periodic domain Ω, the viscous Burgers problem is to find u that solves the following:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial t^2} \tag{4.6.1}$$

$$u(x, 0) = u_0(x) \tag{4.6.2}$$

The ν constant is the diffusivity, or kinematic viscosity, which is chosen based on the phenomenon to be modelled. The viscous Burgers equation is regularised by its diffusive term, which eliminates the possibility of shocks to form and prevents the strong form problem from only having weak solutions. In the case where $\nu \neq 0$, an analytic solution can be found by means of a Cole-Hopf transformation [34] which reduces the nonlinear equation to the linear

heat equation:

$$u(x,t) = -\frac{2\nu}{w}\frac{\partial w}{\partial x} \implies \frac{\partial w}{\partial t} = \nu\frac{\partial^2 w}{\partial t^2}, w(x,0) = w_0(x) \tag{4.6.3}$$

$$\implies u(x,t) = \frac{\int_{-\infty}^{\infty} \exp[-(x-\xi)^2/4\nu t] \exp[-(2\nu)^{-1}\int_0^{\xi} u_0(\eta)d\eta]u_0(\xi)d\xi}{\int_{-\infty}^{\infty} \exp[-(x-\xi)^2/4\nu t] \exp[-(2\nu)^{-1}\int_0^{\xi} u_0(\eta)d\eta]d\xi} \tag{4.6.4}$$

Salih [35] provides an exact solution using the Cole-Hopf transformation by picking the initial condition to be the periodic function:

$$u_0(x) = \frac{2\nu\alpha\kappa sin(\kappa x)}{\beta + \alpha\cos(\kappa x)} \tag{4.6.5}$$

$$\implies u(x,t) = \frac{2\nu\alpha\kappa e^{-\nu\kappa^2 t}\sin(\kappa x)}{\beta + \alpha e^{-\nu\kappa^2 t}\cos(\kappa x)} \tag{4.6.6}$$

Taking a periodic one-dimensional domain $[0, L]$ on the time-interval $[0, T]$, the space-time weak variational problem for the viscous Burgers' equation is to find $u_{kh} \in W_{kh}$ such that for all $v_{kh} \in W_{kh}$:

$$\int_0^L [u_{kh}v_{kh}]_{\partial[0,T]}dx - \int_0^T\int_0^L u_{kh}\frac{\partial v_{kh}}{\partial t}dxdt - \int_0^T\int_0^L \frac{1}{2}u_{kh}^2\frac{\partial v_{kh}}{\partial x}dxdt + \int_0^T\int_0^L \nu\frac{\partial u_{kh}}{\partial x}\frac{\partial v_{kh}}{\partial x}dxdt = 0 \tag{4.6.7}$$

This corresponds to the UFL Fetsome compatible variational forms:

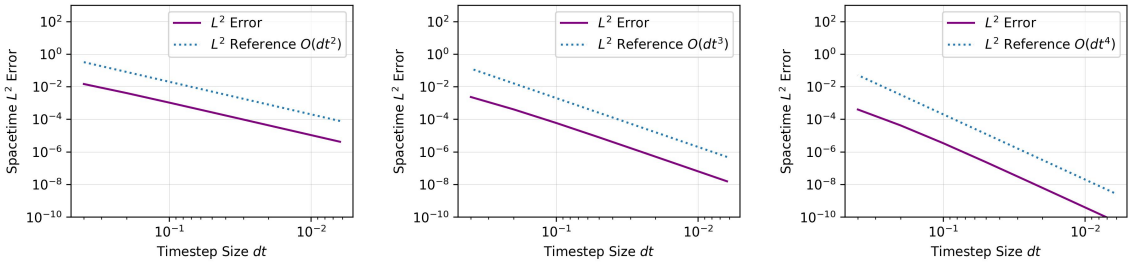
```
b = (-u*Dt(v) - 1./2.*(u**2)*v.dx(0) + nu*u.dx(0)*v.dx(0))*dx
db = u*v*dx
```

For the Fetsome specification, the spatial domain $[0, L] = [0, 8]$ is discretised using degree 3 continuous Lagrange elements. The temporal domain $[0, T] = [0, 0.8]$ is discretised with continuous Petrov-Galerkin elements. The values for the constant parameters are chosen to be:

$$\nu = 0.05, \quad \alpha = 1.5, \quad \beta = 1.55, \quad \kappa = \frac{\pi}{2} \tag{4.6.8}$$

It has been found that 2500 spatial elements yield convergence within 0.1% of the space-time error (see appendix A.4). Figure 4.6 presents a visual confrontation of the numerical and exact solutions and figure 4.7 presents the convergence properties obtained for linear, cubic and quadratic elements.

As can be clearly seen, all required convergence rates are achieved in the asymptotic regime as $\Delta t \to 0$. Therefore, it can be concluded that Fetsome correctly handles the nonlinearity present in the viscous Burgers' equation. For quadratic and cubic elements, an inspection of L^2 error line shows that for time step sizes bigger than $\mathcal{O}(10^{-1})$ the convergence order decreases, indicating that for this order of Δt other numerical errors, such as that of the space discretisation, dominate. Nevertheless, this does not influence the result obtained in the approximate limit as the time step is too large to be considered asymptotic.



(a) Linear time elements (b) Quadratic time elements (c) Cubic time elements

Figure 4.7: L^2 error of viscous Burgers' equation with reference convergence rate lines

4.7 Discontinuous Inviscid Burgers' Equation

The inviscid Burgers' equation, obtained by setting $\nu = 0$ in (4.6.1), is of even greater numerical interest than its viscous counterpart. In the absence of a viscous diffusion term, it is in fact possible for derivative discontinuities to develop and for shock solutions to propagate [36]. Therefore, analytical solutions often cannot be found or involve complicated interactions between shocks and rarefaction fans. Also due to this, it is not possible to analyse the numerical solution using a MMS approach without influencing the regularity of the solution a priori. Instead of considering the convergence orders of the FET solution for this problem, extensively considered in the previous examples, this problem illustrates the capabilities of Fetsome in interfacing with more complex, standard Firedrake spatial discretisations to improve the stability of time stepped solutions.

The inviscid Burgers problem is to find $u(x,t)$ on the spatially-periodic space-time domain $\Omega \times [0,T] = [0,5] \times [0,0.5)$ (with $t = 0.5$ being the time when a smooth solution stops existing for the following $u_0(x)$), such that the following are satisfied:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \tag{4.7.1}$$

$$u(x,0) = u_0(x) = \sin\left(\frac{4\pi x}{5}\right) \tag{4.7.2}$$

For the space-time finite element discretisation, we pick dG elements of degree 3 for the time domain and dG elements of degree 1 for the space domain. The doubly-discontinuous weak variational problem is to find $u_{kh} \in W_{kh}$ such that for all $v_{kh} \in W_{kh}$:

$$\sum_{I_n \subset [0,T]} \sum_{K \subset \Omega} \left[\int_K [u_{kh} v_{kh}]_{\partial I_n} dx - \int_{I_n} \int_K u_{kh} \frac{\partial v_{kh}}{\partial t} dx dt - \int_{I_n} \int_\Omega \frac{1}{2} u_{kh}^2 \frac{\partial v_{kh}}{\partial x} dx dt \right.$$
$$\left. + \int_{I_n} \int_{\partial K} \frac{1}{2} u_{kh}^2 v_{kh} \cdot n ds dt \right] = 0 \tag{4.7.3}$$

As in 2.2.4, the integrals over ∂I_n and ∂K require a choice of numerical flux. While the flux function in the time domain is chosen to be the time upwind flux, more care has to be taken to choose the flux between spatial elements on the ∂K boundaries. As suggested by [19], the optimal choice for a nonlinear problem such as the inviscid Burgers' equation would be

the Godunov flux, but in this case the Lax-Friedrichs flux provides an effective yet simpler approximation to it:

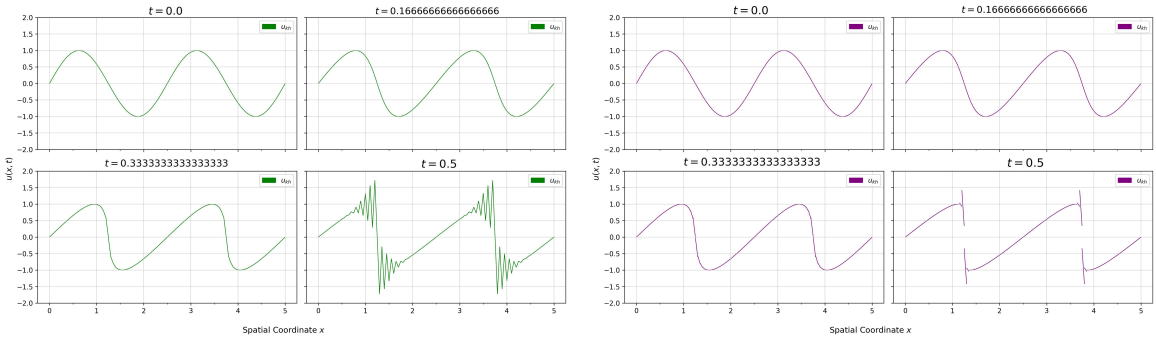
$$\hat{f}(u_{kh}^-, u_{kh}^+) = \frac{1}{2}\left(\frac{1}{2}(u_{kh}^+)^2 + \frac{1}{2}(u_{kh}^-)^2\right) - \frac{c}{2}(u_{kh}^+ n^- + u_{kh}^- n^+) \tag{4.7.4}$$

In the flux function, c is an estimate for what is called the maximum wave speed of the problem. It can be taken to be $c = 1$ for the given initial condition u_0, its maximum value. With this choice of numerical flux, the UFL forms representing the doubly-discontinuous space-time inviscid Burgers' problem to be supplied to Fetsome are:

```
b = (-u*Dt(v) - 1./2.*(u**2)*v.dx(0))*dx
        + (0.5*(0.5*u('+')**2 + 0.5*u('-')**2)
           - 0.5*(u('+')*n('-')[0] + u('-')*n('+')[0]))
        * (n('+')[0]*v('+') + n('-')[0]*v('-'))*dS
db = u*v*dx
```

The doubly-discontinuous discretisation is to be compared with the discretisation involving degree 3 cPG elements in time and degree 1 continuous Lagrange elements in space. Both formulations involve 100 spatial elements and a timestep size of $\Delta t = 0.0625$.

Figure 4.8a illustrates the solution obtained with cPG+cG elements. It is clearly seen that artificial oscillatory behaviour becomes dominant around the regions with a steep derivative as the time approaches $t = 0.5$, the moment when a smooth solution stops existing. This instability of the chosen finite element spaces is unsatisfactory for a meaningful numerical solution of the inviscid Burgers' problem. On the other hand, figure 4.8b presents the dG+dG solution. It is evident that oscillations are absent and the solution presents none of the instabilities present in the cPG+cG choice of space-time elements around the regions of interest. Therefore, it can be concluded that improvements in solution stability can be achieved with the two time elements currently provided by Fetsome and their combination with more complex spatial discretisations from core Firedrake.



(a) cPG + cG space-time discretisation

(b) dG + dG space-time discretisation

Figure 4.8: Comparison of the oscillatory behaviour and its absence for the two space-time discretisations of the inviscid Burgers' problem

Chapter 5

Conclusion and Extensions

5.1 Project Conclusions

This report has presented Fetsome, a new UFL manipulation layer for the specification and automatic solution of finite element in time problems for Firedrake and Irksome. By developing the software abstractions for the mathematics underpinning the theory of FET, the presented implementation gives the capability for the users of these two finite element frameworks to supply UFL forms to a FET time stepping solver, the `VariationalTimeStepper`. Interpreting the supplied forms as space-time integrals, the manipulation layer automatically composes the UFL forms representing the FET problem by coupled finite element problems in space to be passed to the standard Firedrake solver, all with minimal input from the user.

Fetsome supports the numerical discretisation of the time domain using continuous Petrov-Galerkin and discontinuous Galerkin elements for arbitrary polynomial degrees automatically, a functionality which is found to be novel in the landscape and literature of general-purpose finite element PDE solvers. Both discretisations of the time domain also support the inclusion of integrals over one-dimensional timestep boundaries, as well as space-time integrals representing time-dependent forcing terms and time-dependent spatial Dirichlet boundary conditions, all contributions not previously found in the literature. The presented implementation also provides a unified approach in solving linear and nonlinear space-time problems alike, needing no further input from its users. The capabilities of Fetsome which it provides for both classes of problems makes it possible for it to target many important time-dependent weak variational problems from across many areas of applied mathematics. This respects the Firedrake philosophy that the core functionality it provides should be as application-independent as possible and accessible for researchers from diverse fields.

The convergence of both continuous Petrov-Galerkin and discontinuous Galerkin FET discretisations has been analysed in depth for purposes of code verification. It has been discussed that the mathematical theory predicts that degree k cPG and dG solutions to space-time weak variational problems should converge to the exact solutions of the discretised PDEs with an L^2 error in space-time asymptotically scaling as $\mathcal{O}(\Delta t^{k+1})$. This has been verified for important fundamental PDEs, namely the heat equation, the linear advection/transport equation and Burgers' equation in its viscid form. The mixture of boundary terms, forcing functions and spatial boundary conditions used by such examples demonstrates the correctness of the implementation and its interacting parts.

Finally, Fetsome has been shown to easily combine with more complicated finite element discretisations in space, such as for the inviscid Burgers' equation with discontinuous Galerkin elements in space. The stability obtained by the dG+dG space-time element choice further highlights the numerical potential of FET discretisations. Furthermore, Fetsome's success with spatial surface integrals involving numerical flux such as Lax-Friedrichs flux highlights the compatibility with the core functionality of Firedrake and the expressivity of UFL which can be exploited by a typical user of these packages. Overall, this makes Fetsome a successful and valuable addition to the UFL symbolic manipulation layers of Firedrake and Irksome.

5.2 Future Work

Having implemented a foundational finite element in time layer for Firedrake and Irksome, there are many possible extensions that can now be attempted, as well as further work that can be done to enrich the functionality developed in this investigation.

5.2.1 Lagrangian Variational Problems

The first important extension to Fetsome that can now be attempted is that of coupling the FET manipulation layer with a layer for the Lagrangian-dynamical specification of problems of physical relevance. This formulation is based on the principle of least action, which states that the path q taken through space by a particle is the one which minimises the action functional given by integration of the Lagrangian $L(q, \dot{q}, t)$ along the particle's path $q(t)$ [37]:

$$\delta S = 0, \quad S[q] = \int_{t_1}^{t_2} L(q, \dot{q}, t) dt \tag{5.2.1}$$

By calculating the variation of the action, the Euler-Lagrange equations of motion are obtained:

$$\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} = 0 \tag{5.2.2}$$

Many finite element in time discretisations have favourable conservation properties for quantities that are induced by symmetries of the Lagrangian [38], therefore making extending Fetsome in this direction relevant for problems of physical interest.

5.2.2 Mixed Time Function Spaces

Fetsome currently only supports space-time FET problems which discretise all time-dependent variable on the same temporal function space. This is sufficient in the case of time-dependent PDEs which are first order in time, but cannot support higher orders of time dependence. Mixed function spaces in time, just as the spaces implemented in standard Firedrake, present the opportunity for Fetsome to most flexibly be extended in the support of these problems. For example, consider the fundamental problem of the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u, \quad u(x,0) = u_0(x), \quad u'(x,0) = g(x), \quad \nabla u \cdot \mathbf{n}|_{\partial\Omega} = 0 \tag{5.2.3}$$

We can build the space-time finite element problem by taking the solution trial function as $u_{kh} \in W_{kh}$ and the test function $v_{kh} \in W_{kh}$. By introducing the new variable $\sigma_{kh} = \frac{\partial u_{kh}}{\partial t}$ on

the space M_{kh} and its appropriate test function $\omega_{kh} \in M_{kh}$, the second order wave equation can be split into two coupled first order weak variational problems:

$$\forall v_{kh} \in W_{kh} : \quad \int_0^T \int_\Omega \frac{\partial \sigma_{kh}}{\partial t} v_{kh} dx dt + \int_0^T \int_\Omega c^2 \nabla u_{kh} \cdot \nabla v_{kh} dx dt = 0 \tag{5.2.4}$$

$$\forall \omega_{kh} \in M_{kh} : \quad \int_0^T \int_\Omega \sigma_{kh} \omega_{kh} dx dt = \int_0^T \int_\Omega \frac{\partial u_{kh}}{\partial t} \omega_{kh} dx dt \tag{5.2.5}$$

With mixed elements in time, the solution of such a system would be possible by combining the two equations in a single effective-space UFL form in such a way to produce no ambiguity between the trial-test function pairs. Mixed function spaces also introduce an outlook for important and active research based on the stability of the combinations of the function spaces W_{kh} and M_{kh} chosen for each of the trial-test function pairs appearing in the weak variational problem. This makes mixed function spaces a very desirable extension to Fetsome.

5.2.3 UFL Space-Time Support

As has been presented in chapter 3 the way in which a collection of UFL forms is interpreted as a space-time variational problem depends on the user forming a triple of forms and supplying it to a `VariationalTimeStepper`. Nevertheless UFL's role as a DSL for the specification of finite element problems means that a much more natural formulation of FET problems can be obtained by adding UFL extensions to Fetsome to support space-time forms, like Irksome introduces the `Dt` UFL operator. By introducing time domain and time-boundary integrals through the similar measures `dt` and `dI` to the supported `dx`, `ds`, `dS` measures, as well as the introduction of a time variable `t` coupled to a `TimeFunction` class, solving FET problems with Fetsome would be better integrated to the Firedrake, Irksome and UFL environments. For example, the forced heat equation with vanishing Neumann conditions would then be described by the UFL form:

```
F = u*v*dx*dI - u*Dt(v)*dx*dt + dot(grad(u), grad(v))*dx*dt - f*v*dx*dt
```

A modification like this would ultimately improve Fetsome's user API and increase its potential as a common tool for the existing, specialisation-diverse Firedrake user base.

5.2.4 Future Studies in Efficiency

Although Fetsome successfully makes use of the Firedrake caching system to improve the efficiency of space-time solutions, there are mathematical manipulations that can also be applied to affect the speed with which solutions are found.

Fetsome automatically creates quadrature rules that can be used to integrate space-time forms: custom chosen quadratures can be passed to the `VariationalTimeStepper` and down to the UFL manipulation layer. Higher orders of quadrature require a greater number of point evaluations and form systems of equations to be solved that are not diagonal, and so not easily invertible. Decreasing the order of quadrature to schemes that are non-exact can provide lower accuracy but still low-error solutions and transform the derived algebraic systems to ones that are solved with much reduced computational costs, and so a greater efficiency.

Another important tool that can be explored in a FET setting is system preconditioning. When solving a linear system $A\mathbf{u} = \mathbf{f}$, a preconditioner P can be used to define the left and right preconditioned problems [20] respectively:

$$(P^{-1}A)\mathbf{u} = P^{-1}\mathbf{f}, \qquad (AP^{-1})(P\mathbf{u}) = (AP^{-1})\mathbf{y} = \mathbf{f} \quad \text{then} \quad P\mathbf{u} = \mathbf{y} \tag{5.2.6}$$

For some choice of P depending on the system, solving the two preconditioning problems can be much faster than solving the original problem. Fetsome allows the specification of PETSc options to be passed down to the Firedrake solver, including the preconditioning layer. Finding good preconditioners for FET is a potential great source of speedup in the solution of space-time problems. Overall, exploring outlets for increased efficiency is an important opportunity to reduce the computational cost of FET and make FET time stepping more desirable compared to more traditional procedures, such as direct Runge-Kutta stepping.

Bibliography

[1] J. G. Skellam. Random dispersal in theoretical populations. *Biometrika*, 38(1-2):196–218, 1951. doi: 10.1093/biomet/38.1-2.196. URL https://doi.org/10.1093/biomet/38.1-2.196. https://academic.oup.com/biomet/article-pdf/38/1-2/196/679108/38-1-2-196.pdf.

[2] Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. Mcrae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. Firedrake: Automating the finite element method by composing abstractions. *ACM Trans.Math.Softw.*, 43 (3), dec 2016. doi: 10.1145/2998441. URL https://doi.org/10.1145/2998441.

[3] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100): 9–23, Dec 7 2015. URL https://journals.ub.uni-heidelberg.de/index.php/ans/article/view/20553/20092.

[4] Oliver Sander. Dune — the distributed and unified numerics environment. pages 43–61, Dec 18, 2020. URL https://link.springer.com/book/10.1007/978-3-030-59702-3#toc.

[5] Patrick E. Farrell, Robert C. Kirby, and Jorge Marchena-Menendez. Irksome: Automating runge–kutta time-stepping for finite element methods, 2020. 2006.16282.

[6] Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 40(2):1–37, 2014.

[7] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, Sep 10, 2009. doi: 10.1002/nme.2579. URL https://api.istex.fr/ark:/67375/WNG-ZSB761VG-7/fulltext.pdf.

[8] Satish Balay, Shrirang Abhyankar, MarkF Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, WilliamD Gropp, V. 'aclav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, MatthewG Knepley, Fande Kong, Scott Kruger, DaveA May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, JoseE Roman, Karl Rupp, Patrick Sanan, Jason Sarich, BarryF Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. Petsc/tao users manual. Technical report, Argonne National Laboratory, 2021.

[9] firedrakeproject and FEniCS. firedrakeproject/fiat: Fiat: Finite element automatic tabulator. URL https://github.com/firedrakeproject/fiat.

[10] FInAT. Finat/finat: finat: a smarter library of finite elements. URL https://github.com/FInAT/FInAT.

[11] Miklós Homolya, Lawrence Mitchell, Fabio Luporini, and David A. Ham. Tsfc: A structure-preserving form compiler. *SIAM journal on scientific computing*, 40(3):C401–C428, Jan 2018. doi: 10.1137/17M1130642. URL https://arxiv.org/abs/1705.03667.

[12] Tuomas Kärnä, Stephan C. Kramer, Lawrence Mitchell, David A. Ham, Matthew D. Piggott, and António M. Baptista. Thetis coastal ocean model: discontinuous galerkin discretization for the three-dimensional hydrostatic equations. *Geoscientific Model Development*, 11(11):4359–4382, Oct 30, 2018. doi: 10.5194/gmd-11-4359-2018. URL https://search.proquest.com/docview/2126769799.

[13] Daniel R. Shapero, Jessica A. Badgeley, Andrew O. Hoffman, and Ian R. Joughin. icepack: a new glacier flow modeling package in python, version 1.0. *Geoscientific Model Development*, 14 (7):4593–4616, Jul 26, 2021. doi: 10.5194/gmd-14-4593-2021. URL https://search.proquest.com/docview/2555115378.

[14] Thomas H. Gibson, Lawrence Mitchell, David A. Ham, and Colin J. Cotter. Slate: extending firedrake's domain-specific abstraction to hybridized solvers for geoscience and beyond. *Geoscientific Model Development*, 13(2):735–761, Feb 25, 2020. doi: 10.5194/gmd-13-735-2020. URL https://search.proquest.com/docview/2362649241.

[15] Alexandre Ern and Jean-Luc Guermond. *Finite Elements III : First-Order and Time-Dependent PDEs*, volume 74. Springer International Publishing, Cham, Jan 1, 2021. ISBN 9783030573478. doi: 10.1007/978-3-030-57348-5. URL https://library.biblioboard.com/viewer/670152c8-b786-11eb-bb72-0a9b31268bf5.

[16] Philippe G. Ciarlet. *The finite element method for elliptic problems*. Society for Industrial and Applied Mathematics, 2002. ISBN 9780898715149. URL http://portal.igpublish.com/iglibrary/search/SIAMB0000110.html.

[17] A. K. Aziz and Peter Monk. Continuous finite elements in space and time for the heat equation. *Mathematics of Computation*, 52(186):255–274, 1989. doi: 10.2307/2008467. URL http://www.jstor.org/stable/2008467. 12.

[18] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1):50–67, 1947. doi: 10.1017/S0305004100023197. URL https://www.cambridge.org/core/article/practical-method-for-numerical-evaluation-of-solutions-of-partial-differential\-equations-of-the-heatconduction-type/B3230893A53384D418228AB39D41A451.

[19] Bernardo Cockburn and Chi-Wang Shu. Runge-kutta discontinuous galerkin methods for convection-dominated problems. *Journal of scientific computing*, 16(3):173–261, 2001. doi: 10.1023/A:1012873910884.

[20] Ed Bueler. *PETSc for Partial Differential Equations: Numerical Solutions in C and Python*, volume 31. SIAM, Society for Industrial and Applied Mathematics, Philadelphia, 2021. ISBN 9781611976304. URL https://www.gbv.de/dms/tib-ub-hannover/1702985075.pdf.

[21] Ernst Hairer, Gerhard Wanner, and Christian Lubich. *Geometric Numerical Integration : Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 9783540306634. doi: 10.1007/3-540-30666-8. URL https://library.biblioboard.com/viewer/12f60573-bfb0-11ea-ae5c-0a28bb48d135.

[22] Johan Hoffman, Johan Jansson, Rodrigo Vilela de Abreu, Niyazi Cem Degirmenci, Niclas Jansson, Kaspar Müller, Murtazo Nazarov, and Jeannette Hiromi Spühler. Unicorn: Parallel adaptive finite element simulation of turbulent flow and fluid–structure interaction for deforming domains and complex geometry. *Computers fluids*, 80(SI):310–319, Jul 10, 2013. doi: 10.1016/j.compfluid.2012.02.003. URL https://dx.doi.org/10.1016/j.compfluid.2012.02.003.

[23] deal.II. Reference documentation for deal.ii version 9.3.3: The step-23 tutorial program, . URL https://www.dealii.org/current/doxygen/deal.II/step_23.html.

[24] deal.II. Reference documentation for deal.ii version 9.3.3: The step-58 tutorial program , . URL https://www.dealii.org/current/doxygen/deal.II/step_58.html.

[25] hp Fem Group. hp-fem group. URL https://www.hpfem.org.

[26] D. Patterson, A. Monti, C. W. Brice, R. A. Dougal, R. O. Pettus, S. Dhulipala, D. C. Kovuri, and T. Bertoncelli. Design and simulation of a permanent-magnet electromagnetic aircraft launcher, 2005. ID: 1.

[27] Y. Duan, M. Keefe, T. A. Bogetti, and B. Powers. Finite element modeling of transverse impact on a ballistic fabric. *International Journal of Mechanical Sciences*, 48(1):33–43, 2006. doi: https://doi.org/10.1016/j.ijmecsci.2005.09.007. URL https://www.sciencedirect.com/science/article/pii/S0020740305002316. ID: 271483.

[28] Muhammad A. Ashraf, Evgeny V. Morozov, and Krishnakumar Shankar. Flexure analysis of spoolable reinforced thermoplastic pipes for offshore oil and gas applications. 33(6):533–542, 2014. journal: Journal of Reinforced Plastics and Composites; https://doi.org/10.1177/0731684413491442.

[29] Mohaddeseh Mousavi Nezhad, Quentin J. Fisher, Elia Gironacci, and Mohammad Rezania. Experimental study and numerical modeling of fracture propagation in shale rocks during brazilian disk test. *Rock Mechanics and Rock Engineering*, 51(6):1755–1775, 2018. doi: 10.1007/s00603-018-1429-x. URL https://doi.org/10.1007/s00603-018-1429-x. ID: Mousavi Nezhad2018.

[30] Robert W. Howarth, Anthony Ingraffea, and Terry Engelder. Should fracking stop? *Nature*, 477(7364):271–275, 2011. doi: 10.1038/477271a. URL https://doi.org/10.1038/477271a. ID: Howarth2011.

[31] A. Hebala, S. Nuzzo, P. H. Connor, P. Giangrande, C. Gerada, and M. Galea. Improved propulsion motor design for a twelve passenger all-electric aircraft. In - *2021 IEEE Workshop on Electrical Machines Design, Control and Diagnosis (WEMDCD)*, pages 343–348, 2021. ISBN NULL-. doi: 10.1109/WEMDCD51469.2021.9425667. ID: 1.

[32] Benjamin B. Wheatley. Appropriate and ethical finite element analysis in mechanical engineering: Learning best practices through simulation. In *2020 ASEE Virtual Annual Conference Content Access*, Virtual On line, June 2020. ASEE Conferences. https://peer.asee.org/34161.

[33] Kambiz Salari and Patrick Knupp. Code verification by the method of manufactured solutions. Technical report, Jun 1, 2000. URL https://www.osti.gov/servlets/purl/759450.

[34] Julian D. Cole. On a quasi-linear parabolic equation occurring in aerodynamics. *Quarterly of applied mathematics*, 9(3):225–236, Oct 1, 1951. doi: 10.1090/qam/42889. URL https://www.jstor.org/stable/43633894.

[35] A. Salih. Burgers' equation, Feb 18, 2016. URL https://www.iist.ac.in/sites/default/files/people/IN08026/Burgers_equation_viscous.pdf.

[36] J. M. Burgers. *A Mathematical Model Illustrating the Theory of Turbulence*, volume 1 of *Advances in Applied Mechanics*, pages 171–199. Elsevier Science Technology, 1948. ISBN 0123745799. doi: 10.1016/S0065-2156(08)70100-5. URL https://dx.doi.org/10.1016/S0065-2156(08)70100-5.

[37] L. D. Landau and E. M. Lifshitz. *Mechanics: Course of Theoretical Physics, Volume 1*. Butterworth-Heinemann, Oxford, Jan 1, 1976. ISBN 9780750628969.

[38] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, 10:357–514, 2001. doi: 10.1017/S096249290100006X. URL https://www.cambridge.org/core/article/discrete-mechanics-and-variational-integrators/C8F45478A9290DEC24E63BB7FBE3CEB5.

Appendix A

Evaluation Data

A.1 Analytic Heat Equation Convergence Data

Number of Spatial Elements	L^2 Error (Cubic cPG, $\Delta t = 0.0625$)	Error Decrease %
400	1.2616204822622198e-09	Chosen
500	1.2609533238223085e-09	-0.05%

Table A.1: Convergence of lowest timestep L^2 error for the 1D heat equation with exact solution for testing procedure.

	Spacetime L^2 Error		
Δt	Linear cPG	Quadratic cPG	Cubic cPG
0.0625	0.0001298843073086203	5.408811609945753e-07	1.2616204822622198e-09
0.125	0.000519416778055539	4.324728880758648e-06	2.0163651433674935e-08
0.25	0.002075743229848816	3.452387446279155e-05	3.2201165670711505e-07
0.5	0.008272404306194645	0.00027385316487789054	5.113868952346959e-06
1.0	0.03261382812441787	0.002119488919428314	7.947835900843637e-05
2.5	0.18634366343823308	0.02711772088251065	0.002605203998636307

Table A.2: L^2 errors for the 1D heat equation with exact solution.

	Relative Convergence Rates		
Δt	Linear cPG	Quadratic cPG	Cubic cPG
0.0625	Baseline	Baseline	Baseline
0.125	1.9996654821713218	2.9992261382637606	3.998407027470216
0.25	1.9986634714467455	2.9969127834486913	3.9972840892177057
0.5	1.9946787049164083	2.987736267419195	3.989230371205943
1.0	1.9791051878436703	2.9522419655074974	3.958074989774075
2.5	1.9020777872541332	2.7818817202182826	3.80859629948175

Table A.3: Relative convergence rates for the 1D heat equation with exact solution.

A.2 MMS Forced Heat Equation Convergence Data

Number of Spatial Elements	L^2 Error (Cubic dG, $\Delta t = 0.03125$)	Error Decrease %
400	2.0573545813458467e-09	Chosen
500	2.058861203606176e-09	-0.007%

Table A.4: Convergence of lowest timestep L^2 error for the 1D forced heat equation for testing procedure.

	Spacetime L^2 Error		
Δt	Linear dG	Quadratic dG	Cubic dG
0.03125	0.00107442242888461	1.3470049007120087e-06	2.0573545813458467e-09
0.0625	0.004297191642260589	1.0730556131722163e-05	3.284383358603389e-08
0.125	0.017180441782850477	8.51072558400099e-05	5.216671803016542e-07
0.25	0.06858588168135032	0.0006688230106869757	8.211735938511207e-06
0.5	0.2721580073692369	0.0051536044220729165	0.00012644857762326606
1.0	1.0541981626830865	0.03814182220830581	0.0018360271795392132
2.0	3.704545608391724	0.2670090160456395	0.022907481999802387

Table A.5: L^2 errors for the 1D forced heat equation.

	Relative Convergence Rates		
Δt	Linear dG	Quadratic dG	Cubic dG
0.03125	Baseline	Baseline	Baseline
0.0625	1.999832791677082	2.9938978437019474	3.9967601643823816
0.125	1.9993011111651227	2.9875572859958606	3.989435236022311
0.25	1.997144495144824	2.974270444244156	3.9764856591093354
0.5	1.9884609482350168	2.9458854148004425	3.9447197624564314
1.0	1.9536296943346183	2.8877200605428768	3.859964701953044
2.0	1.813150512736781	2.8074427834476054	3.641159564814607

Table A.6: Relative convergence rates for the 1D forced heat equation.

A.3 2D Semi-Periodic Transport Equation Convergence Data

	Spacetime L^2 Error		
Δt	Linear cPG	Quadratic cPG	Cubic cPG
0.03125	0.01344466970355208	1.2130067438108289e-05	1.4967421873156793e-06
0.0625	0.053703496229867205	9.663445576437092e-05	1.55897786168289e-06
0.125	0.21361882667242632	0.0007832012785474081	7.159097585945785e-06
0.25	0.8358368135987186	0.00657597660567314	0.00011236394000093577
0.5	3.0731987655195385	0.06085314852061419	0.001824653775345493
1.0	9.235451607025453	0.6315188551006861	0.03185391056491163
2.0	18.005063255460616	5.146298853791152	0.6837294510447999

Table A.7: L^2 errors for the 2D semi periodic transport equation.

	Relative Convergence Rates		
Δt	Linear cPG	Quadratic cPG	Cubic cPG
0.03125	Baseline	Baseline	Baseline
0.0625	1.9979817014451775	2.993950113629163	0.058774701451161704
0.125	1.9919508809685174	3.018773530086016	2.1991773041224
0.25	1.9681825017741321	3.0697501413744823	3.972257562554153
0.5	1.8784478720133868	3.210054837716668	4.021371718108193
1.0	1.5874414304459317	3.3754219943218606	4.125775850320304
2.0	0.9631482532108279	3.0266375251549666	4.42388316767253

Table A.8: Relative convergence rates for the 2D semi periodic transport equation.

A.4 Viscous Burgers' Equation Convergence Data

Number of Spatial Elements	L^2 Error (Cubic cPG, $\Delta t = 0.00625$)	Error Decrease %
2500	5.956239709710478e-11	Chosen
2600	5.956029666467673e-11	-0.003%

Table A.9: Convergence of lowest timestep L^2 error for the 1D viscous Burgers' equation.

	Spacetime L^2 Error		
Δt	Linear cPG	Quadratic cPG	Cubic cPG
0.00625	4.168349955479895e-06	1.5578051430524667e-08	5.956029666467673e-11
0.0125	1.667058007601192e-05	1.2449484726605626e-07	9.509886900420065e-10
0.025	6.663734064673439e-05	9.918729417640358e-07	1.5094780729711198e-08
0.05	0.00026583932849720033	7.81111452158456e-06	2.344832722386521e-07
0.1	0.0010525612592314664	5.9171728290899064e-05	3.4047426899258154e-06
0.2	0.004063612285896424	0.00040575807073770854	4.208993440479101e-05
0.4	0.01457979242158248	0.0023216903731941275	0.0003971785781358016

Table A.10: L^2 errors for the 1D viscous Burgers' equation.

	Relative Convergence Rates		
Δt	Linear cPG	Quadratic cPG	Cubic cPG
0.00625	Baseline	Baseline	Baseline
0.0125	1.9997559959273192	2.998499340315138	3.9970053384545206
0.025	1.9990265212115583	2.99406929258349	3.9884778064113644
0.05	1.996151821837603	2.9773011833231986	3.9573633010478164
0.1	1.9852777417623049	2.9213077162523637	3.8599888607470145
0.2	1.9488585586043459	2.77763985188123	3.6278575840940284
0.4	1.843135515045395	2.5164838872612934	3.2382406466509432

Table A.11: Relative convergence rates for the 1D viscous Burgers' equation.