An Adaptive Finite Element Method Approach to the Heat Equation and the Black-Scholes Equation

G14SCD

MSc Dissertation in

Scientific Computation

2017/2018

School of Mathematical Sciences

University of Nottingham

Thabo Miles 'Matli

Supervisor: Dr. Kris Van Der Zee

I have read and understood the School and University guidelines on plagiarism. I confirm that this work is my own, apart from the acknowledged references.

Abstract

The abstract of the report goes here. The abstract should state the topic(s) under investigation and the main results or conclusions. Methods or approaches should be stated if this is appropriate for the topic. The abstract should be self-contained, concise and clear. The typical length is one paragraph.

Contents

1 Introduction

The origin of the Finite Element Method (FEM) is generally agreed to be a paper by Courant [**?**] in 1943. Though initially obscure, it gained widespread usage in engineering as computing power became more cheaply available. Since then it has become increasingly more common in the natural sciences and more recently in financial industry [**?**]. Though the Finite Difference Method is still overwhelming used to price options the FEM is now sometimes being instead.

Though more technical than the Finite Difference Method, under certain circumstance the FEM has stark advantages. Two notable advantages are that the FEM is simpler to use for Partial Differential Equations (PDEs) with irregular shaped domains and that there is a very well understood theory of aposteriori errors. This theory of errors, which only requires knowledge of the estimated solution, allows for the FEM to be adapted during implementation. This adaptive methodology leads to a solution where errors are guaranteed to be within certain tolerances which allows the user to analyse the effects of changing parameters. Also, adaptivity leads to a solution that should be in some sense efficient as ideally maximum accuracy achieved for the minimum degrees of freedom. This concept of efficiency is what we look to investigate further in this paper.

The pioneering work of Babuska et al [**?**] in the 1980s showed the first examples of how an aposteriori error estimate could be used to implement adaptive FEM. The research moved quickly and attempts at adaptive mesh refinement for parabolic PDEs began towards the end of the same decade see [**?**], [**?**] and others. Despite the research into these methods and the adoption of adaptive FEM in science and engineering there are still some theoretical results outstanding. Convergence and optimal complexity have only been shown for linear elliptical PDEs and only quite recently [**?**], [**?**]. Even these recent results only show that there is convergence to a solution and do not imply an order of convergence for the adaptive methods.

The rest of this dissertation comprises:

...

As show by Bergam [**?**] in their article.

The end of the introductory section would typically outline the structure of the report.

In this template, section **??** gives the background of the topic, sections **??** and **??** contain the bulk of the work and section 8 summarises and discusses what has been achieved. Appendix B displays the raw data, and certain technical calculations for section **??** are deferred to appendix E.

2 The Linear Heat Equation

The Heat Equation provides a simple situation within which we can understand and demonstrate the implementation of the Finite Element Method and adaptive algorithms. Conversely it is fundamental enough that it would allow someone reading this dissertation to easily build on our findings to access a different but connected problem for example the Black-Scholes equation. It can be thought of as a prototypical parabolic equation and this was our motivation for studying it.

In this section we will describe the derivation of the Heat Equation in one dimension. We will first do this in the steady state i.e. where $\frac{du}{dt} = 0$ and then extend this to the time dependent equation. We do this just to refresh the reader's knowledge of the properties of the equation.

2.1 The Heat Equation in Steady State

The steady state equation in one dimension can be thought of as describing a thin rod of uniform material on the interval $I = [0, \mathrm{L}]$. As we are only considering the one dimensional problem heat is only conducted in the x direction. The rod is heated by a source f which we assume to have been acting continuously for long enough to reach the steady state.

Let q be the heat flux i.e flow of energy per unit of area per unit of time and let S be the cross section of the rod. As flux is a vector quantity it needs a direction and we take this as the direction of x increasing. The first law of thermodynamics on conservation of energy tells us that the flow out of the rod must equal the flow in from the heat source. Hence we have:

$$q(L)S(L) - q(0)S(0) = \int_I f dx \tag{2.1}$$

We now divide both sides of (2.1) by L and take $\mathrm{L} \to 0$. Hence we have the differential equation:

$$(Sq)' = f \tag{2.2}$$

Employing Fouriers law which in this context can be understood as the flux being

negatively proportional to the temperature gradient:

$$q = kT' \tag{2.3}$$

here k represents the heat conductivity of the rod.

Together (2.2) and (2.3) give us the Heat Equation:

$$-(SkT')' = f \tag{2.4}$$

2.2 The Time Dependent Problem

The time dependent problem is very similar to the steady state form. We introduce a function e which is the energy per unit length within the rod. We use the the conservation of energy principle again but this time we use the fact that the sum of the resultant heat flux is equal to the rate of change of internal energy. Here rate of change of e is denoted by $e\dot{\text{ff}}$l

$$\int_I \dot{e}dx = q(0)S(0) - q(L)S(L) \int_I f dx \tag{2.5}$$

Once again dividing by L and letting L $\to$ 0.

$$\dot{e} + (Sq)' = f \tag{2.6}$$

Finally assuming that the energy depends linearly on the the Temperature

$$e = mT \tag{2.7}$$

And using (2.3) and combining (2.5) and (2.6) we have the transient Heat Equation which we will refer through out the rest of this paper simply as the Heat Equation:

$$m\dot{T} - (SkT')' = f \tag{2.8}$$

2.3 Boundary Conditions

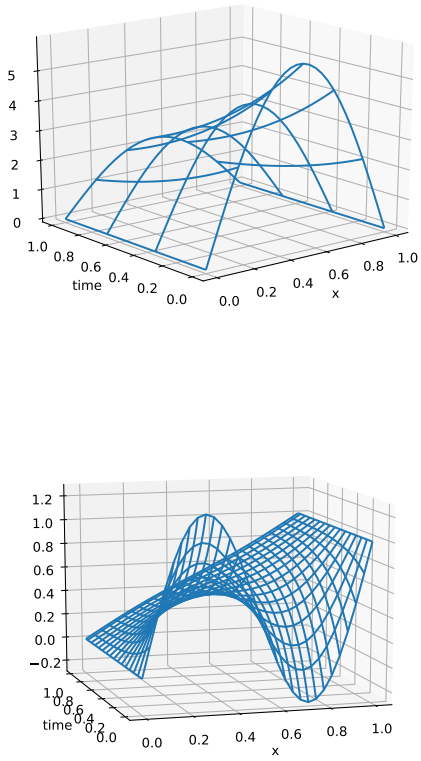


Figure 1: $f = 0$, $m = 1$, $S = 1$, $k = \pi^{-}2$

3 The Finite Element Method

This section is concerned with the discretisation of the elliptical Heat Equation by Finite Elements. Although the focus of this dissertation is adaptive solvers for the time dependent Heat Equation many aspects of the time dependent problem are the same, or only slightly different, from more easily described stationary one. We will therefore introduce a bulk of concepts in this chapter to avoid introducing this content alongside the specific content for the parabolic problem.

Though not uncommon, the FEM is not necessarily widely known by the average postgraduate mathematician. As such we will provide a fairly detailed description of the method and a few of its attendant concepts. A very nice introduction to the subject if more detail is needed is [**?**].

The underlying practical steps to the method can be reduced to finding a variational form of the equation and then solving that form of the equation in an approximated sense. We will add the detail to this as we introduce the relevant background.

The remainder of this chapter will introduce some function spaces and the basis function that we will use for our approximation 3.1, we will then introduce the weak formulation and approximation of an elliptical model problem 3.2, an analysis of the model problem will be carried out in **??** and then finally we will analyse the way we have designed our data structures and why in 3.4.

Finding the variational form:

1. Finding a variational form of the equation we are looking to solve. This will allow us to look for a solution which satisfies the equation in some weak sense. This step requires some knowledge of function spaces and some definitions that we will introduce when necessary.

2. Creating the subdivision of our domain Ω.

3. Taking the variational form, which we have at first defined in an infinite dimensional function space V, and approximating it on a finite dimensional sub-space V_h. In our case the subspace will be piecewise polynomial functions defined on our subdivision.

4. Projecting the boundary conditions onto the finite dimensional space.

Once the mathematical context of the Finite Element has been outlined we will move on to finding the weak formulation of the Heat Equation. With this done we will consider the finite dimensional subspace and the elements that will be used in our discretisation. We will close the chapter will a discussion of other possible choices for elements.

3.1 Supporting Theory

We begin by stating the notation to be used in this dissertation and also stating some of the relevant definitions and results from functional analysis.

3.1.1 Notation and the Weak Derivative

Definition 3.1. Continuous Functions Let $\Omega \subset \mathbb{R}^n, n \geq 1$ be an open bounded set.

- We define $C(\Omega)$ the set of all real-valued and continuous functions defined on Ω.
- For $m \geq 1$ we write $C^m(\Omega)$ to denote the set of m times continuously differentiable functions i.e. that $C^m(\Omega) = \{f \in C(\Omega) : f^{(k)} \in C(\Omega) \quad \forall k \leq m\}$.
- Let $C_0^\infty(\Omega)$ denote the set of all infinitely many times differentiable that vanish on the boundary of Ω.

The multi-index is a notational tool which allows for the simplification of statements in multi-dimensional calculus and is useful for our descriptions of PDEs. It is specifically an array α which carries information about our partial derivatives.

Definition 3.2. Multi Index

$\alpha = (\alpha_1, \alpha_2, ...\alpha_n) \in \mathbb{N}^n \qquad |\alpha| = \alpha_1 + ...\alpha_n$
$D^\alpha = \left(\frac{\partial}{\partial x_1}\right)^{\alpha 1} ... \left(\frac{\partial}{\partial x_n}\right)^{\alpha_n} = \frac{\partial^{|\alpha|}}{\partial x_1^\alpha ... \partial x_n^\alpha}$
$C^m(\Omega) = \{f \in C(\Omega) : D^\alpha f \in C(\Omega) \quad \forall |a| \leq\leq m\}$

The FEM makes extensive use of piecewise-linear functions. These functions may not have derivatives in the traditional continuous sense but we may generalise the concept of a derivative to admit certain piecewise functions.

Definition 3.3. Weak Derivative

Let u be a locally integrable function on Ω. Say there exists a function ψ_α which is also locally integrable on ω such that:

$\int_\Omega \psi_\alpha(x) \cdot v(x)dx = (-1)^\alpha \int_\Omega u \cdot D^\alpha v \forall v \in C_0^\infty(\Omega)$

Then ψ_α is the weak derivative of u of order $|a|$ hence $\psi_\alpha = D^\alpha u$.

3.1.2 Function Spaces

To find a suitable weak formulation we need to take some results from the theory of function spaces.

Definition 3.4. L_2 Space Let $L_2(\Omega)$ denote the set of all real valued functions defined on Ω with $\Omega \subset \mathbb{R}^n$ such that:

$$\|u\|_{L_2(\Omega)} := \Big(\int_\Omega |u|^2 dx \Big)^{\frac{1}{p}} < \infty$$

Definition 3.5. Sobolev Space

$H^m(\Omega) = \{u \in L_2(\Omega) : D^\alpha u \in L_2(\Omega), |\alpha| \leq m\}$

$$\|u\|_{H^m(\Omega)} := \Big(\sum_{|\alpha| \leq m} \|D^\alpha u\|^2_{L_2(\Omega)} \Big)^{\frac{1}{2}}$$

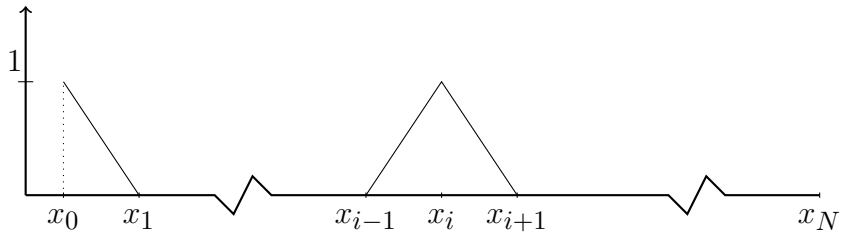
3.1.3 Basis Functions

Take the interval $x \in [0, L]$ and the intervals $[x_k, x_{k+1}] \quad k = 0, 1, ..., N-1$ with $h_i = x_{i+1} - x_i$. Then we can define the below function which is referred to a the FEM basis function or the "hat" function.

$$\phi_j(x) = \Big(1 - \Big| \frac{x - x_k}{h_i} \Big| \Big)_+, \quad j = 0, 1, ..., N \tag{3.1}$$

The interval leads to two half hats on the boundary. Clearly the basis function does not have a continuous derivative but it does have a weak derivative in the sense described above 3.3.

The function space through which we will choose to view this function



The Lax Milgram theorem shows shows the existence and uniqueness of a solution to the PDE.

3.2 An Elliptical Model Problem

Now we have defined the mathematical context let us solve a problem via the FEM. Take the stationary heat equation.

$$-u'' = f \qquad x \in [0,1] \tag{3.2a}$$

$$u(0) = 0 \tag{3.2b}$$

$$u(1) = 0 \tag{3.2c}$$

The first step of solving this and any PDE by the FEM is to find a suitable weak formulation. This is given by multiplying both sides of the equation by a test function in the Sobolev space $v \in H_0^1(0, L)$.

$$\int_0^L u''v dx = \int_0^L fv dx \qquad dx \forall v \in H_0^1(0, L)$$

Integrating by parts on the left leads us to our weak formulation of 3.2

Problem 3.1. *Weak Formulation*

Find $u \in H_0^1(0, 1)$*:*

$$\int_0^L u'v' dx = \int_0^L fv dx \qquad \forall v \in H_0^1(0, L)$$

$\forall v \in H_0^1(0, L)$

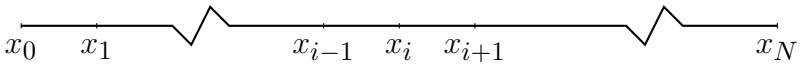
Define

$$\int_0^L u'v' dx = a(u, v) \int_0^L fv dx = \ell(v)$$

This gives us the linear and bilinear form with can be used to show by A that u exists and is unique. However, our test function has been chosen simply some function in the Sobolev space which has infinitely many functions defined within it and is described as infinite dimensional.

To find the FEM approximation to **??** we look for a solution in a finite dimensional subspace of $V \subset H^1(0, L)$. The subspace we chose consists of continuous piecewise linear functions of defined on some given subdivision of Ω. Then we can restate the weak formulation by the following:

Take the subdivision on $x \in \Omega$ in our model problem 3.2. We subdivide the domain into N intervals $[x_k, x_{k+1}] \quad k = 0, 1, ..., N-1$. This subdivision may not be regular and when later in this dissertation we begin seeking an adaptive solution will very rarely be regular. intervals may be referred to as elements and give the FEM its name.



Problem 3.2. *FEM Approximation Find* $u_h \in V \subset H^1(0,1) \quad : \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \subset V.$

Now we can define our approximation u_h to 3.2 by:

$$u_h(x) = \sum_{j=1}^{N(h)} U_j \cdot \phi_h(x)$$

With $U_j \; j = 1, ..., N(h)$ being constants to be determined. We can now rewrite 3.2 by the following:

$$\sum_{j=1}^{N(h)} U_j \, a(\, \phi_j(x), \, \phi_i(x)) = \ell(\, \phi_i(x)\,)$$

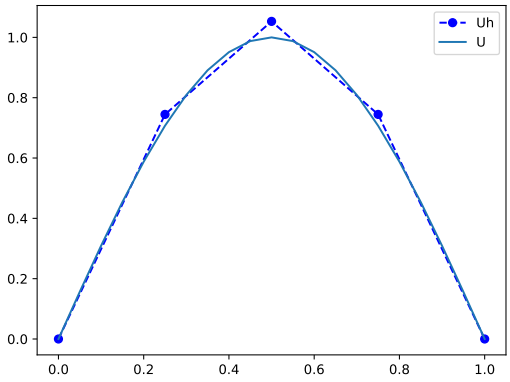
This gives a system of linear equations with $a(\, \phi_j(x), \, \phi_i(x))$ an $N(h) \times N(h)$ matrix.

$$A_{ij} = \int_I \phi_i' \phi_j' dx \qquad b_i = \int_I f \phi_i dx$$

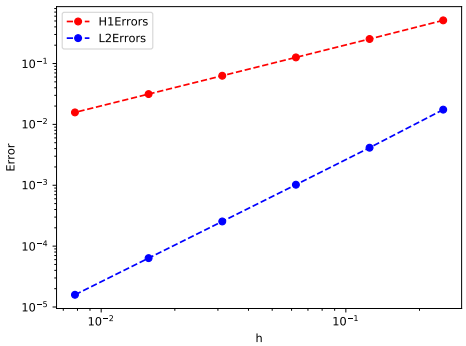
Hence the system $AU = F$

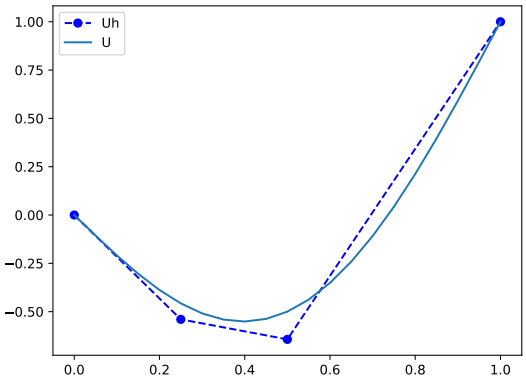
3.3 Numerical Experiments

First we take 3.2 with right-hand side function $\pi^2 \cos(\pi x - 0.5\pi)$ and a uniform mesh $x \in [0, 0.25, 0.5, 0.75, 1]$.

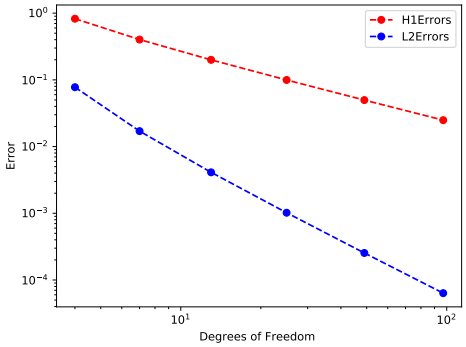


h	$\|u-u_h\|_{L_2(0,1)}$	$\|u-u_h\|_{H^1(0,1)}$
$\frac{1}{4}$	0.0174706	0.511854
$\frac{1}{8}$	0.00413879	0.252837
$\frac{1}{16}$	0.00102063	0.12604
$\frac{1}{32}$	0.000254283	0.0629727
$\frac{1}{64}$	6.3516e-005	0.0314805





DOGs	$\|u-u_h\|_{L_2(0,1)}$	$\|u-u_h\|_{H^1(0,1)}$
4	0.0776354	0.825144
7	0.0170398	0.40185
13	0.00411272	0.199548
25	0.00101902	0.0996014
49	0.000254182	0.0497791



3.4 Implementation

It is evident that all the matrices that arise from the one dimensional FEM are tridiagonal. This is important as there are efficient algorithms available for solving tridiagonal systems. To store the matrix we only need to none zero arrays that compose it. This is an ideal opportunity to create a data structure to store these matrices and also to make use of some object oriented design patterns to associate the characteristics of the matrices to a special case of tridiagonal matrices such as **??**. One possible declaration is given below:

```
class TriDiagMatrix{
public:
    void SetMatrix( std::vector<double> Diagonal,
         std::vector<double> LowerDiag, std::vector<double> UpperDiag);

    void MatrixSolver( std::vector<double> f,
         std::vector<double> &x );

protected:
    std::vector<double> mpDiagonal;
    std::vector<double> mpLowerDiag;
    std::vector<double> mpUpperDiag;
};
```

The obvious advantage of a similar declaration is not only the efficiency of the storage and the methods but also the readability, polymorphism and the ease with which the code can then be tested. Once module tests have been run and effectiveness established we can derived any specific matrices we need from the above interface. The first derivation we make is for **??** also called the Stiffness matrix. This class reads a specific subdivision of the domain ω and sets the Stiffness matrix accordingly.

```
class StiffnessMatrix: public TriDiagMatrix{
public:
    void BuildStiffnessMatrix ( SpaceMesh& smesh ); };
```

We are now ready to solve **??**. We can now write a solve method which is as simple

as the simplest psuedo-code. Called for 3.2 with $f = \pi^2 cos(\pi x - \frac{\pi}{2})$, subdivision of Ω so that $h_i = 0.25i = 1, 2, 3, 4$ the code produces the below result.

```
void StationaryHeatEquation(){
StiffnessMatrix ( mpsmesh );
buildfvec( SpaceMesh& a_smesh, double(*f)(double))
stiff.MatrixSolver( f_vec, U );
PrintVector(U);
}
```

We have not yet discussed the implementation of boundary conditions. In the derivation of the weak formulation the boundary conditions are projected into the solution space. This has been done in 3.1 by choosing $H_0^1(0, L)$ for our zero dirichlet boundaries.

In practise one of the major advantages of the FEM is the ease with which more complicated boundary conditions can be implemented. A detailed account of doing this in a general is given in [**?**]. The method they advocate is implementing Robin boundary conditions and using parameters to set Neuman conditions or approximate dirichlet conditions. From here onward this is the method used in the code from here on.

4 The Time Dependent Problem

Now that we have built some foundations by solving the elliptical PDE refeq:Steady Diffusion1 we can use the same method to discretise the space dimension of the Heat Equation.

$$\dot{u} - u'' = f \quad x \in [0, L], \quad t \in (0, T] \tag{4.1a}$$

$$u(0, t) = u(L, t) = 0 \tag{4.1b}$$

$$u(x, 0) = u_0(x) \tag{4.1c}$$

Multplying $\dot{u} - u'' = f$ by a test function $v \in H_0^1(0, L)$ on both sides gives:

$$\int_0^L \dot{u}v dx - \int_0^L u''v dx = \int_0^L fv dx$$

Integrating by parts yields:

$$\int_0^L \dot{u}v dx + \int_0^L u''v + u(0)v(0) - u(L)v(L) dx = \int_0^L fv dx$$

$$\int_0^L \dot{u}v dx + \int_0^L u'v' dx = \int_0^L fv dx$$

The weak formulation of our problem is now given by:

Problem 4.1. *Weak Formulation Find* $u(\cdot, t) \in H_0^1(0, L)$*:*

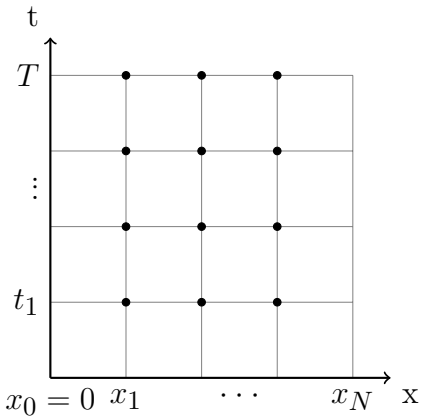
$$(\dot{u}(\cdot, t), v) + a(u(\cdot, t), v) = (f(\cdot, t), v)$$
$$(u(\cdot, 0), v) = (u_0, v)$$

$\forall v \in H_0^1(0, L)$

Here,

$$a(w, v) = \int_0^L w'v' dx (w, v) = \int_0^L wv dx$$

To find our approximation define a mesh on $\bar{\Omega}_h^{\Delta t} = [0, 1] \times [0, T]$.



Let the partition on the space mesh be $0 = x_0 < x_1 < ... < x_N = 1$ with intervals $h_i = x_i - x_{i-1}$. Let the time partition be $0 = t_0 < t_1 < ... < t_M = T$ time steps $\Delta t_j = t_j - t_{j-1}$. For both space and time we allow the intervals to vary.

Let $V \subset H_0^1(0,1)$ denote the set of all continuous piecewise linear functions defined on the space mesh that are zero on the boundaries $x = 0 and x = 1$.

4.0.1 Implicit Euler

4.1 is the a space discrete problem, sometimes referred to as the semi-discreet problem but we now need to discretise in time to be able to find an approximation to the solution. We will use Finite Differences to discretise in time and we will choose the Implicit Euler method.

Problem 4.2. *Implict Euler Method*
Find $u_h^{m+1} \in V \quad 0 \leq m \leq M-1$

$$\begin{aligned} \Big(\frac{u_h^{m+1} - u_h^m}{\Delta t}, v_h\Big) + a(u_h^m, v_h) &= (f(\cdot, t^m), v_h) \\ (u_h^0 - u_0, v_h) &= 0 \quad \forall v_h \in V \end{aligned}$$

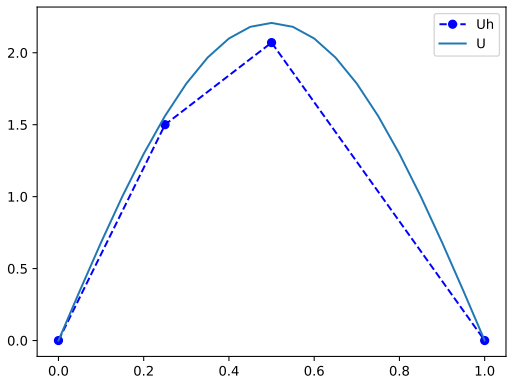
We can therefore rewrite the system as:

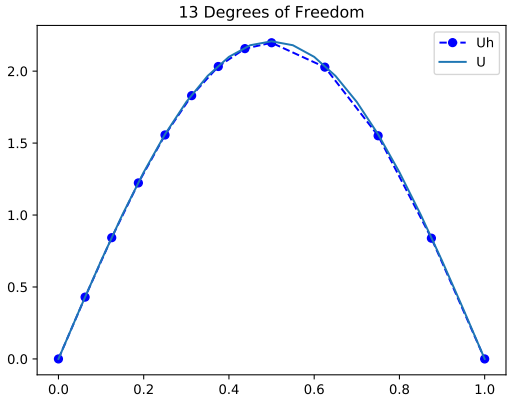
$$\begin{aligned} (u_h^{m+1}\, v_h) + \Delta t a(u_h^m, v_h) &= (u_h^m,\, v_h) + \Delta t (f(\cdot, t^m), v_h) \\ v_h \quad \forall v_h \in V \quad 0 \leq m \leq M-1 \end{aligned}$$

with

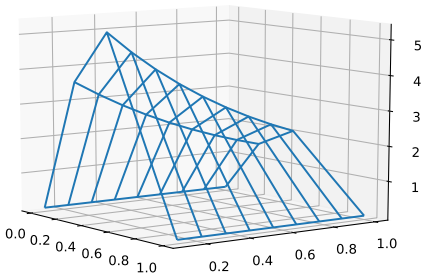
$$(u_h^0) \quad = \quad u_0, v_h \quad \forall v_h \in V \tag{4.2}$$

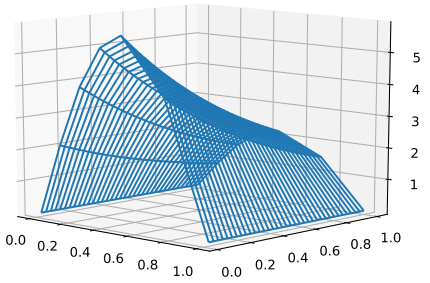
4.1 Results





$Mesh$	DOGs	$\|u-u_h\|_{L_2(0,1)}$	s	$\|u-u_h\|_{H^1(0,1)}$	s
1	4	0.304466	2	1.74657	5
2	7	0.0918245	5	0.883062	11
3	13	0.0231464	5	0.439949	11
4	25	0.00575675	2	0.219779	5
5	49	0.00143142	5	0.109867	11
6	97	0.000356574	5	0.054931	11

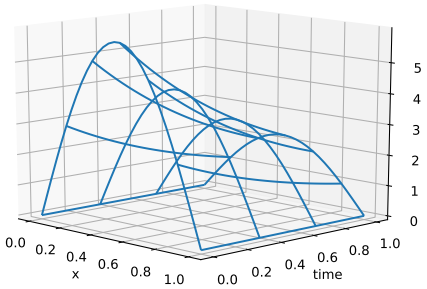


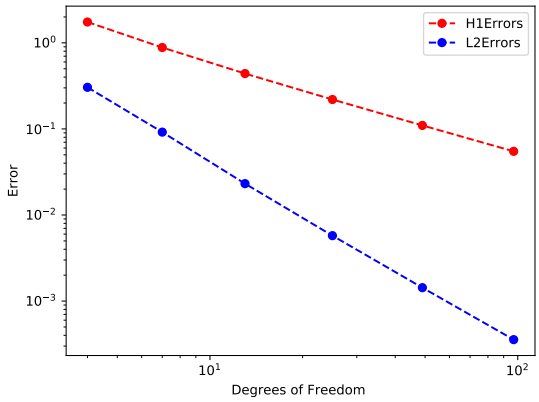


4.2 Implementation

When it comes to the computer implementation of the FEM on time dependent or multidimensional problems "meshing" is the important art-form. There are entire software packages available to carry out the creation and manipulation of mesh structures and it would therefore benefit any FEM project to either design there own data-structures or implement an existing mesh package.

An important question when designing the data-structure and class inheritance structure is the relationship between the time-steps and the space nodes. When using Finite Difference Methods to discretise the time element the space nodes are free to vary at at a given time step however the time steps are the same when considered from a given node. This seems to both invite and discourage the creation of a single data-structure to handle meshing in both space and time. Much of the functionality of each is shared but much differs:






```
class SpaceMesh
{
public:
    void CopySpaceMesh (const SpaceMesh& oldSpaceMesh);
    void BisectIntervals (std::vector<int> &intervalsForBisection);
    int meshsize();

    void CommonMesh( SpaceMesh& firstmesh, SpaceMesh& secondmesh );
    double TestFunctions(int nodeIndex, double x);
    bool Contained (double my_var );

protected:
    std::vector<double> mpSpaceNodes ;
    std::vector<double> mpSpaceMesh;
    int mpmeshsize;
};
```

```
class TimeMesh
{
public:
    void CopyTimeMesh (const TimeMesh& oldTimeMesh);
    void GenerateTimeMesh ( std::vector<double> TimeNodes );
    void BisectInterval (int lowerIndex, int upperIndex);
    int NumberOfTimeSteps();
protected:
    int mpnumberOfTimeSteps;
    std::vector<double> mpTimeNodes;
    std::vector<double> mpTimeMesh;
};
```

The question of how to structure inheritance in FEM meshes is in this case restricted to the one dimension and may seem abstract at the beginning of a project. However, there are many utility style functions i.e. adding a node or copy construction which could be simplified if there was an abstraction which could be justified rigorously. The danger of a heuristic abstraction pattern is that one of the child classes have access to a function which is not truly polymorphic and it cannot be used safely or that the program not scale because of restrictions on the inheritance. We leave the question of whether there is a rigorous way to argue for a shared parent class for a space and time meshes open.

5 A Posteriori Error Analysis

When a continuous problem such as the Heat Equation is discretised Finite Element Methods and solved numerically an error is incurred. Some knowledge about the error is available in advance of the implementation of a specific FEM approximation. This type of error analysis, so called a priori, is valuable for the analysis of convergence and other proofs that may be important about an FEM scheme in general. Although useful in these applications it is rare that an a priori error estimate accurately quantifies the error present in a numerical solution. As such in a practical context the a priori error cannot be used to indicate the quality of a solution.

In contrast, a posteriori error analysis uses the computed numerical solution itself to analyses the error. Where a posteriori error estimates are available and accurate this has the obvious advantage that it describes the error in the approximation without the need for an analytical solution. This replaces the heuristics and guess work that had previously guided the error analysis of the FEM approximation in settings where no analytical solution was available.

Since the initial pioneering methods of Babuska [**?**] [**?**] in this field many a posteriori error methods have been presented. Many of these have been shown to be effective in context. The emphasis in the literature has moved now from the discovery of new a posteriori estimates to the testing of the limitations of existing a posteriori estimates. All this is done primarily with the goal of validating the extensively used self-adapting FEM packages.

5.0.1 Gradient Recovery

The FEM provides an approximation to an unknown function, however the gradient of the function may also be of particular interest also. The gradient in the approximation is discontinuous over element boundaries but can be smoothed out by post-processing. In fact under certain circumstances the smoothed gradient is closer to the gradient of the true solution [**?**]. This leads fairly directly one of the simpler a posteriori error estimates which is measuring the gradient before and after this post-processing.

Consider again the steady state one dimensional Heat Equation with conductivity

co-efficient of one 3.2 and here restating its weak formulation for ease.

Problem 5.1. *Weak Formulation*

Find $u \in H_0^1(0,1)$*:*

$$\int_0^L u'v'dx = \int_0^L fvdx \qquad dx \forall v \in H_0^1(0,L)$$

$\forall v \in H_0^1(0,L)$

With

$$\int_0^L u'v'dx = a(u,v)$$

It is clear that the bilinear form is symmetric i.e. a(w, v) = a(v, w) and it can be shown that it is coercive as described in the Lax Milgram theorem A. Hence, $a(w,v)$ is an inner product on $H_0^1(0,L)(w,v)_a$.

We can then define the associated energy norm.

$$\|w\|_a = |(w,w)_a|^{\frac{1}{2}} \tag{5.1}$$

If we measure the true error in the energy norm we get.

$$\|u - u_h\|_a^2 = \int_0^L u' - u_h' dx \tag{5.2}$$

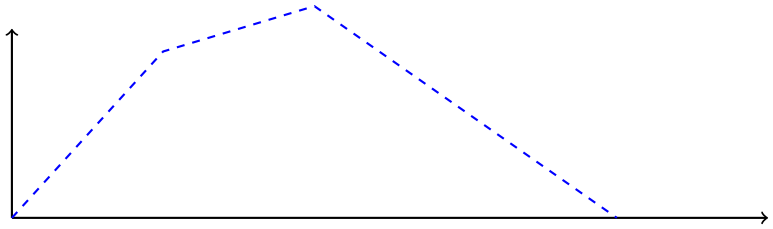
Which is the L_2 norm of of the difference between the gradient .

If we had the true gradient we would now be able to measure the true error in the energy norm. As that is unavailable we will use an estimate of the true gradient which will be obtained by some suitable post- processing of the FEM approximation to the solution. Let the post processed FEM approximated gradient be denoted by $G_h(u_h)$ then the error indicator for **??** is given by:

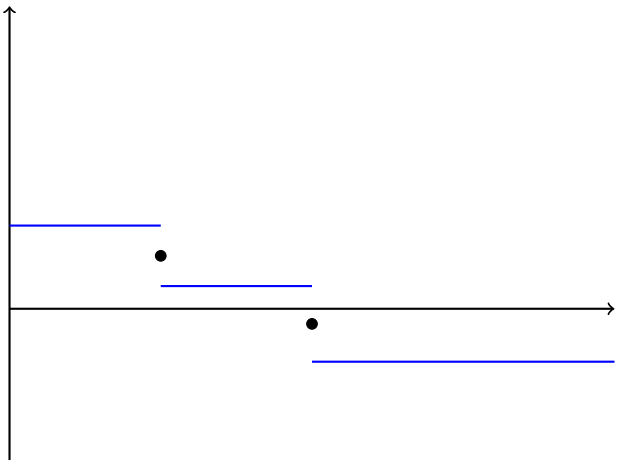
$$\eta^2 = \int_0^L |G_h(u_h) - u_h|^2 dx \tag{5.3}$$

Clearly there are as many different error indicators as there are ways of processing the FEM approximation to the gradient.

Consider an FEM approximation to an PDE of form 4.1b at an arbitrary time step.



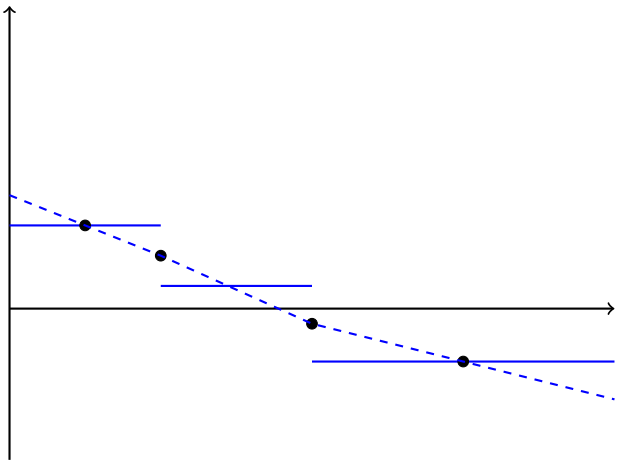
The gradient of this approximation is a discontinuous step-like function, undefined at the nodes of the space mesh.



Intuitively a better approximation would be given by a function defined in the whole domain. A simple but logical one to use could be a continuous piecewise linear function as we have the architecture in our code to linearly interpolate sets of points. We take the midpoint of the FEM approximated gradient where currently our gradient is undefined and hence where we expect post-processing to have the greatest effect.

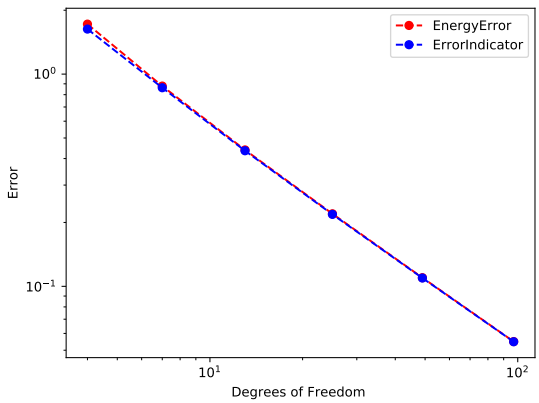
More formally results exist to support this choice. In the case of a uniform partition the "super-convergence" of the centroid of two elements is shown by [**?**]. Though true "super-convergence" may not be assured on our more general partition our method none-the-less should be an improvement on the untreated gradient.

A we also need to decide what to do with the elements at each side as our current method will not produce points for on the boundaries. We will choose to have the interpolant take the value of the gradient in the centre of the elements.

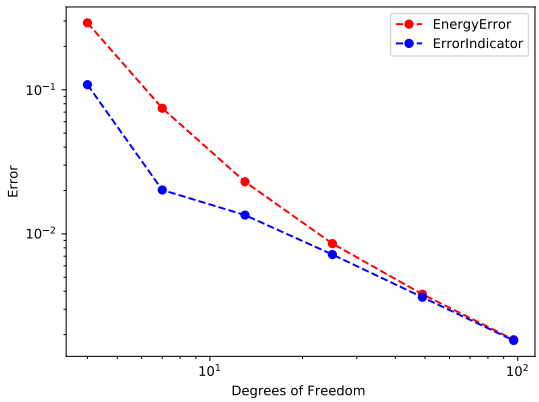


5.1 Results

Mesh	DOGs	$\|u - u_h\|_{a(0,1)}$	$\|\eta\|$
1	4	1.63002	1.71983
2	7	0.862629	0.878275
3	13	0.435436	0.43934
4	25	0.218313	0.219703
5	49	0.109433	0.109858
6	97	0.0548128	0.0549298



$Mesh$	DOGs	$\|u - u_h\|_{a(0,1)}$	$\|\eta\|$
1	4	0.29131	0.108467
2	7	0.074447	0.0201663
3	13	0.0229916	0.0135003
4	25	0.00855775	0.00718688
5	49	0.00381445	0.00363495
6	97	0.00184453	0.00182187



5.2 Implementation

To make the most use of our existing member functions in the SpaceMesh class we should make sure our gradient recovery vector match the space nodes. This will allow the interpolant to be completely defined by the new vector and a reference to a grid. This means that the first and last point though should be defined on the boundary. In practise this means hard coding an interpolant over the first and last element and taking its value at the boundary.

```
void GeneralHeat::GradientRecoveryFunction( SpaceMesh& relevantMesh,
       std::vector<double>& gradvec, std::vector<double>& gradrecovery ) {

        //(x_0, y_0) (x_1, y_1) define the line segment to be hard coded
    double x_0 = 0.5*(relevantMesh.ReadSpaceNode(1)+
                                relevantMesh.ReadSpaceNode(0));
```

```
    double y_0 = gradvec.at(0);
    double x_1 = relevantMesh.ReadSpaceNode(1);
    double y_1 = 0.5*(gradvec.at(1)+gradvec.at(0));

    gradrecovery.push_back(y_0+(relevantMesh.ReadSpaceNode(0)-
                                                x_0)*(y_1-y_0)/(x_1-x_0));

                //take the midpoint of the two gradients where undefined
    for(int i = 0; i<relevantMesh.meshsize()-1; i++)
    {
        gradrecovery.push_back(0.5*(gradvec.at(i)+gradvec.at(i+1)));
    }

                //the hardcoding needs to happen on both sides
    x_0 = relevantMesh.ReadSpaceNode(mpsmesh.meshsize()-1);
    y_0 = gradrecovery.back();
    x_1 = 0.5*(relevantMesh.ReadSpaceNode(relevantMesh.meshsize())
    +relevantMesh.ReadSpaceNode(relevantMesh.meshsize()-1));
    y_1 = gradvec.back();

    gradrecovery.push_back(y_0+
    (mpsmesh.ReadSpaceNode(mpsmesh.meshsize())-x_0)*(y_1-y_0)/(x_1-x_0));
}
```

6 Design of Adaptive Algorithms

6.0.1 Space Adaptivity

6.1 Results

6.2 Time Adaptivity

7 Some Numerical Examples

8 Conclusions

Further help on LaTeX can be found easily on the internet. The LaTeX wikibook[1] contains a lot. For instance you would find there how to type theorems and proofs nicely. Or how to include source code written in some programming language like matlab. There are long lists available with all sorts of common mathematical symbols like ξ, ∇, ∞, log, $\Longleftrightarrow$, etc.

[1] http://en.wikibooks.org/wiki/LaTeX

A Lax Milgram

Theorem A.1 (Lax Milgram). *Let V be a real Hilbert Space with associated norm $\|\dot{\text{ffl}}\|$. Also let $a(\dot{\text{ffl}}, \dot{\text{ffl}})$ be a bilinear functional on V x V with:*

(a) $a(\cdot,\cdot)$ is coercive, i.e. there exists a positive constant $c_0 \forall v \in V | a(v,v)| \geq c_0 \|v\|_V^2$

(b) $a(\cdot,\cdot)$ is continuous i.e. there exists a positive constant $c_1 \forall v, w \in V | a(v,w) \leq c_1 \|v\|_V \|w\|_V$

and for $\ell(\cdot)$ a bilinear functional also on V such that

(c) $\ell(\cdot)$ is continuous i.e. there exists a positive constant $c_2 \forall v \in V | \ell(\cdot)| \leq c_2 \|v\|_V$

Then there $\exists u \in V : a(u,v) = \ell(v) \forall v \exists V$ and u is unique.

B Tridiagonal Matrices

```
class TriDiagMatrix
{
public:

    void SetMatrix( std::vector<double> Diagonal,
        std::vector<double> LowerDiag, std::vector<double> UpperDiag);

    void MatrixVectorMultiplier ( std::vector<double> f, std::vector<double> &Produc

    void PrintMatrix();

    void MatrixSolver( std::vector<double> f,
         std::vector<double> &x );

    void MultiplyByScalar (double k);

    void AddTwoMatrices ( TriDiagMatrix firstMatrix, TriDiagMatrix secondMatix );

protected:

    int mpn;
```

```
    std::vector<double> mpDiagonal;
    std::vector<double> mpLowerDiag;
    std::vector<double> mpUpperDiag;

};

void TriDiagMatrix::MatrixVectorMultiplier ( std::vector<double> f, std::vector<doub
    {
        ProductVector.assign( mpn, 0 );

        ProductVector[0] = mpDiagonal[0]*f[0]+mpUpperDiag[0]*f[1];
        ProductVector[mpn-1] = mpLowerDiag[mpn-1]*f[mpn-2]+mpDiagonal[mpn-1]*f[mpn-1

            for (int j = 1; j<mpn-1; j++)
            {
                ProductVector[j] = mpLowerDiag[j]*f[j-1]+mpDiagonal[j]*f[j]+mpUpperD
            }
    }

    void TriDiagMatrix::MatrixSolver( std::vector<double> f, std::vector<double> &x
    {
        std::vector<double> cDiagonal = mpDiagonal;
        std::vector<double> cLowerDiag = mpLowerDiag;
        std::vector<double> cUpperDiag = mpUpperDiag;
        x.assign(mpn, 0.);
         //scaling matrix entries
    for(int i=1; i<mpn; i++)
    {
        cDiagonal.at(i) = cDiagonal.at(i)-(cUpperDiag.at(i-1)*cLowerDiag.at(i)/cDiag
        f.at(i) = f.at(i)-(f.at(i-1)*cLowerDiag.at(i)/cDiagonal.at(i-1));
    }

        //solving through elimination
     x.at(mpn-1)=f.at(mpn-1)/cDiagonal.at(mpn-1);
     for(int i=mpn-2; i>=0; i=i-1)
     {
```

```
        x.at(i) = (f.at(i) - cUpperDiag.at(i)*x.at(i+1))/(cDiagonal.at(i));
    }
   }
```

C Stiffness and Mass Matrices

```
class StiffnessMatrix: public TriDiagMatrix
{
public:

    void BuildStiffnessMatrix ( SpaceMesh smesh );
    void BuildGeneralStiffnessMatrix ( SpaceMesh smesh );
    void SetParameters (double k_0, double k_L, double constant);

protected:
    const double M_PI = 2*acos(0);
    double a, mpk_0, mpk_L;
};

void StiffnessMatrix::BuildGeneralStiffnessMatrix ( SpaceMesh smesh )
{
mpn = smesh.meshsize()+1;
mpDiagonal = { a*pow(smesh.ReadSpaceMesh(0), -1)+mpk_0 };
mpLowerDiag = {0};
mpUpperDiag = { -1*a*pow( smesh.ReadSpaceMesh(0), -1 ) };

for(int i=0; i<mpn-2; i++)
    {
        mpDiagonal.push_back( a*pow(smesh.ReadSpaceMesh(i), -1) + a*pow(smesh.ReadSp
        mpLowerDiag.push_back( -1*a*pow(smesh.ReadSpaceMesh(i),-1)  );
        mpUpperDiag.push_back( -1*a*pow(smesh.ReadSpaceMesh(i+1), -1) );
    }

mpDiagonal.push_back( a*pow(smesh.ReadSpaceMesh(mpn-2), -1)+mpk_L );
mpLowerDiag.push_back( -1*a*pow(smesh.ReadSpaceMesh(mpn-2),-1) );
```

```
mpUpperDiag.push_back( 0 );
}

class MassMatrix: public TriDiagMatrix
{
public:
    void BuildMassMatrix ( SpaceMesh smesh );
    void BuildGeneralMassMatrix ( SpaceMesh smesh );
};

void MassMatrix::BuildGeneralMassMatrix ( SpaceMesh smesh )
{
mpn = smesh.meshsize()+1;
mpDiagonal = {smesh.ReadSpaceMesh(0)*pow(3,-1)};
mpLowerDiag = {0};
mpUpperDiag = {(smesh.ReadSpaceMesh(0)*pow(6,-1))};

for(int i=0; i<mpn-2; i++)
    {
        mpDiagonal.push_back( (smesh.ReadSpaceMesh(i)+smesh.ReadSpaceMesh(i+1))*pow(
        mpLowerDiag.push_back( smesh.ReadSpaceMesh(i)*pow(6,-1) );
        mpUpperDiag.push_back( smesh.ReadSpaceMesh(i+1)*pow(6,-1) );
    }

mpDiagonal.push_back( smesh.ReadSpaceMesh(mpn-2)*pow(3,-1)) ;
mpLowerDiag.push_back( smesh.ReadSpaceMesh(mpn-2)*pow(6,-1) );
mpUpperDiag.push_back( 0 );
}
```

D Space and Time Meshes

```
class SpaceMesh
{
public:
    void Range( double lowerlimit, double upperlimit, std::vector<double>& Nodes);
    void CommonMesh( SpaceMesh& firstmesh, SpaceMesh& secondmesh );
```

```
    double GeneralTestFunctions(int nodeIndex, double x);
    bool Contained (double my_var );

    void CopySpaceMesh (const SpaceMesh& oldSpaceMesh);

    void BisectIntervals (std::vector<int> &intervalsForBisection);
    void CoarsenIntervals (std::vector<int> &intervalsForCoarsening);

    void GloballyBisectSpaceMesh ();


    int IndexAbove ( double x );


protected:
    std::vector<double> mpSpaceNodes;
};
```

E Calculations for section ??

In this appendix we verify equation (??).