Adaptive hp-Finite Element Methods

G14DIS = MATH4001

Mathematics 4th Year Dissertation

2021/22

School of Mathematical Sciences

University of Nottingham

Marcus Blowers

Supervisor: Dr. Edward Hall

Assessment type: Review

Abstract

The aim of this project was to code and analyse the efficacy of an adaptive hp finite element method in solving a 1 dimensional linear differential equation with dirichlet boundary conditions. In order to do this, a general hp finite element method with a suitable piecewise polynomial approximation was implemented and demonstrated to be a valid model. An adaptive algorithm was then designed such that, using a error estimate based on the residual of the differential equation posed, the model could automatically be refined to suit its local behaviours across the domain. This complete model was then analysed and it was determined that, although a global refinement method was more efficient for low accuracy approximations or particularly smooth functions, the adaptive finite element method was extremely effective and efficient for high accuracy and badly behaved function solutions.

Contents

1 Introduction

The hp adaptive finite element method is a powerful numerical method used to approximate differential equations with high accuracy and efficiency, particularly solutions with complicated behaviours or geometries. It combines the strengths of both the finite element method and adaptive mesh refinement techniques to achieve accurate approximations with relatively minimal computational power. This makes it a very sought after method in scientific and engineering applications where problems being solved may not be smooth or defined on convenient domains and therefore cannot be well approximated by more conventional methods such as the finite difference method [20] or spectral methods [22].

Initially created to resolve complex structural engineering problems, the finite element method was developed through the work of mathematicians like Courant [4]. This method divides a continuous domain into a mesh of sub-domains called 'elements' on which the model to a given problem can be approximated as a polynomial of a constant order on each element. This technique was then developed further by the work of mathematicians like Szabo and Babuska [3],[2] into the hp finite element method which solves partial differential equations using piecewise polynomial approximations on a grid of elements where the width h_k and polynomial degree p_k can be varied across the domain.

The hp finite element method is now set up in such a way that each element can be refined locally to the individual behaviour of the system on each of these intervals. Therefore, an adaptive method can be derived which locally refines the values h_k and p_k at each kth element to iterate towards a final approximation of the whole model. This is known as the adaptive hp finite element method. It will be shown in section 14 of this report that, particularly for the complicated solutions mentioned, this is a much more efficient scheme than a global adaptive scheme or either of its predecessors

It is the aim of this project to write code to execute this algorithm in a mathematically rigorous way and prove its efficacy as applied to a one-dimensional advection-diffusion problem with dirichlet boundary conditions.

In order to apply the finite element method, some relevant function spaces as well as the concept of a 'weak derivative' will be established in section 2. Having established this, section 3 will set up the model problem being solved and manipulate it into what is known as its 'weak state'. This allows the incorporation of the function spaces from section 2 which allow certain results to be utilised later.

Some work will then be done in section 4 to show that setting the problem up in this way results in a well posed problem (i.e. one where a unique solution exists).

There is now sufficient basis to apply the finite element method to the problem established in section 3 to be the second order elliptic equation with dirichlet boundary conditions as mentioned previously. In section 5, the details of this method will be explained, with the model for the solution function being established as a linear combination of controlled basis functions. Substituting this combination back into the weak form of the problem yields a system of equations that can be resolved to recover the full approximation model.

A further proof will then be carried out in section 6 to show that solving the problem as established in 5 results in a sensible approximation to the solution.

Having verified this, the finite element method can be explicitly applied at linear order to the problem in section 7. From here, the model will be built up hierarchically to a second order model in section 8 before being generalised to a pth order approximation in section 9. In order to rigorously prove that these implementations are being carried out correctly, their orders of convergence will be compared to the known theory in section 10

These methodologies will allow a generalised approximation to be made in section 11 using any mesh of element widths with given orders at each. This will lay the groundwork for an adaptive algorithm which will be established in section 13.

In order to be able to excecute an adaptive algorithm, however, appropriate error estimates will need to be implemented so that it is known on which elements refinements to the model need to be made. The details of the estimates used will be covered in section 12.

An adaptive method will then be able to be established in section 13 to iterate towards a locally adaptive model.

The algorithm will be then be run on a suitable problem in section 14 and the outputs compared to the known solution as well as other algorithms' approximations and conclusions will be drawn about the efficacy of the model based on these comparisons.

2 Definition of function spaces

Defined here are several function spaces which can be used to formalise definitions and utilise properties in later steps. Their importance may not be clear immediately but as the report progresses they will be implemented in a meaningful way.

The domain of the functions being dealt with is given as $x \in [a, b]$

2.1 Continuously differentiable function spaces $C^k([a, b])$

Definition 2.1. $\underline{C^k([a, b])}$

$C^k(\Omega) = \{u : \Omega \to \mathbb{R} \mid \frac{\partial^\alpha u}{\partial x^\alpha}$ exists and is continuous $\forall \alpha \leq k$ where $\alpha, k \in \mathbb{N}_0 \cup \{\infty\}\}$

(i.e. the functions 'u' in this space are k-times continuously differentiable)

2.1.1 Subspaces

A further set is defined based on this one as:

Definition 2.2. $\underline{C^k_{D(A,B)}([a, b])}$

$$C^k_{D(A,B)}([a, b]) = \{u(x) : [a, b] \to \mathbb{R} \mid u(x) \in C^k([a, b])$$
$$, u(a) = A, u(b) = B \text{ where } k \in \mathbb{N}_0 \cup \{\infty\}\}$$

from which the specific case:

Definition 2.3. $\underline{C^k_0([a, b])}$

$$C^k_0([a, b]) = C^k_{D(0,0)}([a, b]) = \{u(x) : [a, b] \to \mathbb{R} \mid u(x) \in C^k([a, b]), u(a) = u(b) = 0\}$$

is also derived.

2.2 Finitely integrable function spaces $L^p(\Omega)$

$L^p([a, b])$ is the set of real valued functions for which the L^p norm is finite. i.e:

Definition 2.4. $\underline{L^p([a, b])}$

$L^p([a, b]) = \{u : [a, b] \to \mathbb{R} \mid ||u||_{L^p([a,b])} < \infty\}$

where the L^p norm is defined:

$||u||_{L^p([a,b])} = \int_a^b |u|^p dx^{\frac{1}{p}} dx$

2.3 Sobolev function spaces $H^m([a,b])$

Sobolev spaces are spaces defined based on the properties of the so called 'weak derivative' of a function.

2.3.1 Weak derivative

The concept of the weak derivative can be derived using the function spaces established in (2.4) and (2.3):

Let $u \in C^k([a,b])$ for some $k \in \mathbb{N}_0 \cup \{\infty\}$ and $v \in C_0^\infty([a,b])$ (as defined in (2.3)).

Then integration by parts yields:

$\int_a^b u'v dx = [uv]_a^b - \int_a^b uv' dx$

utilising the boundary conditions of $C_0^\infty([a,b])$ gives:

$\int_a^b u'v dx = - \int_a^b uv' dx$

Repeating this process k times yields:

$\int_a^b u^{(k)} \cdot v dx = (-1)^k \int_a^b u \cdot v^{(k)} dx$

This gives the framework by which the weak derivative is defined:

Definition 2.5. Weak Derivative

For $u \in L^2([a,b])$ if $\exists D_\alpha(u(x)) \in L^2([a,b]) s.t.$

$$\int_a^b D_\alpha(u(x)) \cdot v(x) dx = (-1)^\alpha \int_a^b u(x) \cdot v^{(\alpha)}(x) dx \qquad \forall v \in C_0^\infty([a,b])$$

Then the α^{th} weak derivative of u is $D_\alpha(u(x))$

This is a generalisation of the concept of a derivative where the weak derivative of a function can now be defined even if said function doesn't have a classical derivative. All that is required is that the function is finitely integrable as in (2.4).

2.3.2 Sobolev space definition

The Sobolev space is a well defined function space with the following general prescription:

Definition 2.6. General Sobolev space

$W_p^m([a,b]) = \{u \in L^p([a,b]) | D_\alpha(u(x)) \in L^p([a,b]) \text{ for } \alpha \leq m\}$

With the specific case where $p = 2$ being defined separately as:

Definition 2.7. Sobolev space

$H^m([a,b]) = W_2^m([a,b]) = \{u \in L^2([a,b]) : [a,b] \to \mathbb{R} | D_\alpha(u(x)) \in L^2([a,b])$ for $\alpha \leq m\}$

This is the set of square integrable functions (2.4) which is m times weakly differentiable (2.5) and whose weak derivatives are also square integrable.

The space has an associated norm:

Definition 2.8. Sobolev norm

$||u||_{H^m([a,b])} = (\sum_{\alpha \leq m} ||D_\alpha(u(x))||^2_{L^2([a,b])})^{\frac{1}{2}}$

2.3.3 Additional Sobolev spaces

A space is defined to satisfy some generic dirichlet boundary conditions:

Definition 2.9. $H^m_{D(A,B)}([a,b])$

$H^m_{D(A,B)}([a,b]) = \{u \in H^m([a,b]) : [a,b] \to \mathbb{R} | u(a) = A, u(b) = B\}$

From this, a specific subset is defined similarly to (2.3) with 0 as the boundary conditions:

Definition 2.10. $H^m_0([a,b])$

$H^m_0([a,b]) = H^m_{D(0,0)}([a,b]) = \{u \in H^m([a,b]) : [a,b] \to \mathbb{R} | u(a) = u(b) = 0\}$

3 Model problem

The model problem is a one-dimensional advection-diffusion equation with dirichlet boundary conditions which should allow for a wide range of possible systems to be solved for.

3.1 Differential equation form

The form of equations being solved for will be given as:

$$\begin{aligned}&\underline{\text{Model problem}}\\&\text{For } m, n \in \mathbb{R} \text{ and some function } f(x) : [a, b] \to \mathbb{R}\\&-u''(x) + m \cdot u'(x) + n \cdot u(x) = f(x) \qquad \forall x \in (a, b)\\&\text{Subject to boundary conditions: } u(a) = A, u(b) = B\end{aligned} \tag{3.1}$$

(Note that, for the problem to be well defined, the classical solution should satisfy $u \in C^2_{D(A,B)}([a, b])$ for the first and second derivatives to be continuous)

3.2 Converting the problem to its weak form

An essential part of the finite element method is converting the strong form shown in (3.1) to a so called 'weak form'.

This is a rearrangement of the equation where new functions are introduced that can be used to more readily manipulate the equations and spaces involved in the solution.

3.3 Weak form derivation

The 'strong form' of this problem is the problem as stated in (3.1) where u is being solved for:

$$\begin{aligned}&\underline{\text{Strong form}}\\&\text{Find } u(x) \in C^2_{D(A,B)}([a, b]) \text{ s.t.}\\&-u^{''}(x) + m \cdot u'(x) + n \cdot u(x) = f(x)\end{aligned} \tag{3.2}$$

3.3.1 Weak form derivation for (3.2)

From the strong form defined above, the following manipulations can be made:

1. Multiply by some test function $v(x)$ in the function space $H_0^1([a,b])$ such that the system becomes:

   $-u^{''}(x)v(x) + m \cdot u'(x)v(x) + n \cdot u(x)v(x) = f(x)v(x)$

2. Integrate this over the domain:

   $\Rightarrow -\int_a^b u^{''}(x)v(x)dx + m\int_a^b u'(x)v(x)dx + n\int_a^b u(x)v(x)dx = \int_a^b f(x)v(x)dx$

   Integrating the first equation by parts then gives:

   $-([v(x)u^{'}(x)]_a^b - \int_a^b v^{'}(x)u^{'}(x)dx) + m\int_a^b u'(x)v(x)dx + n\int_a^b u(x)v(x)dx = \int_a^b f(x)v(x)dx$

3. By the definition of the function space $H_0^1([a,b])$ as defined in 2.10, $v(a) = v(b) = 0$ which implies:

   $\Rightarrow [v(x)u^{'}(x)]_a^b = 0$

   This simplifies the system to:

Weak form of problem (3.2)

Find $u(x) \in H^1_{D(A,B)}([a,b])$ s.t. $\forall v(x) \in H_0^1([a,b])$: (3.3)

$$\int_a^b v^{'}(x)u^{'}(x)dx + m \cdot \int_a^b u^{'}(x)v(x)dx + n \cdot \int_a^b u(x)v(x)dx = \int_a^b f(x)v(x)dx$$

Which is the desired weak form of the problem.

3.3.2 Weak form motivation

This has relaxed the conditions for the solution. Instead of requiring that the function needs to have continuous second derivatives, the only conditions are now that the function and its weak derivative have to be finitely integrable. This means that this can solve to a system which doesn't even have a well defined derivative. This allows for a wider range of possible solutions including piecewise functions and functions with discontinuities which would not be admitted in the strong form. This builds the framework for a simpler method of analysis.

4 Proving the problem is well posed

4.1 Well-posedness

For a reliable solution to be given to the problem, it is generally required that the problem is 'well-posed'. This is given as in Strauss (2008,p25) [19] as the following conditions:

$$\begin{aligned}&1.\text{There exists at least one solution } u \text{ solving the differential equation}\\&2.\text{This solution is unique}\\&3.\text{The solution depends in a stable manner on the input data}\\&\text{(i.e. for small changes on the input, the solution only changes a little)}\end{aligned} \tag{4.1}$$

4.2 Lax-Milgram Theorem

The Lax-Milgram theorem is a theorem which can be used to prove the existence of a unique solution to equations of linear functionals. Crucially, this can be applied to the system that has already been established. If proved valid, it would ensure the first 2 conditions of 4.1 were satisfied and therefore the weak form would be a well posed problem to which a solution could always be found.

4.2.1 Prerequisites

Linear functionals

Definition 4.1. A linear functional on a real vector space V is defined as a function T on a real vector space V subject to:

$$\{T : V \to \mathbb{R} | T(\alpha v_1 + \beta v_2) = \alpha T(v_1) + \beta T(v_2) \forall \alpha, \beta \in \mathbb{R}, \forall v_1, v_2 \in V\}$$

Bilinear forms

Definition 4.2. A bilinear form on a real vector space V is defined as a function B on the real vector space $V \times V$ subject to:

$$1)\{B : V \times V \to \mathbb{R} | B(\alpha w_1 + \beta w_2, v) = \alpha B(w_1, v) + \beta B(w_2, v) \forall \alpha, \beta \in \mathbb{R}, \forall v, w_1, w_2 \in V\}$$

$$2)\{B : V \times V \to \mathbb{R} | B(w, \alpha v_1 + \beta v_2) = \alpha B(w, v_1) + \beta B(w, v_2) \forall \alpha, \beta \in \mathbb{R}, \forall w, v_1, v_2 \in V\}$$

4.2.2 Lax-Milgram Theorem

The following is a widely known result, the statement and proof of which can be found in Evans (2010) p315 [12].

Definition 4.3. Lax-Milgram Theorem

Suppose V is a real Hilbert space (e.g. H^1) with associated norm: $||\cdot||_V$

Let $a(\cdot,\cdot)$be a bilinear functional on $V \times V$ and $l(\cdot)$ be a linear functional on V.

Then, if the following are true:

$$\begin{aligned} &1)\ \exists c_0 > 0 \text{ s.t. } \forall v \in V \qquad a(v,v) \geq c_0||v||_V^2 \qquad \text{(coercivity in a)} \\ &2)\ \exists c_1 > 0 \text{ s.t. } \forall v,w \in V \qquad |a(w,v)| \leq c_1||w||_V||v||_V \qquad \text{(continuity in a)} \\ &3)\ \exists c_2 > 0 \text{ s.t. } \forall v \in V \qquad |l(v)| \leq c_2||v||_V \qquad \text{(continuity in l)} \end{aligned} \tag{4.2}$$

Then there exists a unique $u \in V s.t.$

$a(u,v) = l(v) \qquad \forall v \in V$

4.3 Writing the weak form as a bi-linear problem

A simple rearrangement of 3.3 gives an equivalent form of the weak formulation as:

Initial bilinear form of the weak formulation 3.3

Find $u(x) \in H^1_{D(A,B)}([a,b])$ s.t. $\forall v(x) \in H^1_0([a,b])$:

$$a(u,v) = l(v) \tag{4.3}$$

where $a(u,v) = \int_a^b v'(x)u'(x)dx + m \cdot \int_a^b u'(x)v(x)dx + n \cdot \int_a^b u(x)v(x)dx$

and $l(v) = \int_a^b f(x)v(x)dx$

So, if it can be proved that the $a(u,v)$ defined above is a bilinear functional and the conditions of the Lax-Milgram theorem are met then it can be concluded that there is always going to be a unique solution to the weak formulation.

In order for the coercivity condition of the Lax-Milgram theorem to be met however, an alternative rearrangement is sought.

Definition 4.4. Rearrangement of u

Let $u(x) = u_0(x) + u_D(x)$

where $u_0(x) \in H_0^1([a,b])$, $u_D(x) \in H^1_{D([a,b])}([a,b])$

(i.e. the function $u_D(x)$ has been created exclusively to satisfy the boundary conditions, the remaining system is solved by u_0)

Under this prescription, the system becomes:

$a(u,v) = a(u_0 + u_D, v) = l(v)$

$\Rightarrow a(u_0, v) = l(v) - a(u_D, v)$ by the bi-linearity of a (shown explicitly in 4.5.1)

where $l(v) - a(u_D, v)$ can be regarded as a a linear functional $\hat{l}(v)$ as u_D is just fixing the boundary conditions so can be set to a simple known function and therefore isn't being solved for in the system.

Therefore, the new statement of the problem in its bilinear form is given:

Rearrangement of the weak formulation bilinear form 4.3

Find $u_0(x) \in H_0^1([a,b])$ s.t. $\forall v(x) \in H_0^1([a,b])$:

$$a(u_0, v) = \hat{l}(v)$$

$$\text{where } a(u,v) = \int_a^b v'(x)u'(x)dx + m \cdot \int_a^b u'(x)v(x)dx + n \cdot \int_a^b u(x)v(x)dx \tag{4.4}$$

and $\hat{l}(v) = l(v) - a(u_D, v)$

for $l(v) = \int_a^b f(x)v(x)dx$

4.4 Inequalities

The following proof require the use of some known inequalities which are given here to be used later.

4.4.1 Poincaré-Friedrichs inequality

Lemma 1. Poincaré-Friedrichs inequality

Let $v \in H_0^1([a,b])$. Then:

$\int_a^b |v|^2 dx \leq \frac{1}{2}(b-a)^2 \int_a^b |v'|^2 dx$

which can be proved with the following steps:

$$\begin{aligned}
\int_a^b |v|^2 dx &= \int_a^b \left| \int_a^x 1 \cdot v'(\xi) d\xi \right|^2 dx \text{ (By the fundamental theorem of calculus)} \\
&\leq \int_a^b \left(\int_a^x |1|^2 d\xi \int_a^x |v'(\xi)|^2 d\xi \right) dx \\
&\leq \int_a^b \left((x-a) \int_a^x |v'(\xi)|^2 d\xi \right) dx \\
&\leq \int_a^b \left((x-a) \int_a^b |v'(\xi)|^2 d\xi \right) dx \\
&= \int_a^b (x-a) dx \int_a^b |v'(\xi)|^2 d\xi \\
&= \frac{1}{2}(b-a)^2 \int_a^b |v'(\xi)|^2 d\xi
\end{aligned} \tag{4.5}$$

4.4.2 Cauchy-Schwarz inequality

Lemma 2. Cauchy-Schwarz inequality

$$|\langle u(x), v(x) \rangle| = | \int_a^b u(x)v(x)dx| \leq \left(\int_a^b u(x)^2 dx \right)^{\frac{1}{2}} \cdot \left(\int_a^b v(x)^2 dx \right)^{\frac{1}{2}} = ||u||_{L^2([a,b])} ||v||_{L^2([a,b])}$$

This is a well known result, the proof of which is omitted but can be found in Poole (2014, p. 539-540) [14]

4.5 Applying Lax-Milgram to the weak formulation

The form is now set up such that the Lax-Milgram theorem can be proved:

4.5.1 Proof that $a(\cdot,\cdot)$ is bilinear

$$
\begin{aligned}
1)a(\alpha u_1+\beta u_2,v) &= \int_a^b v^{'}(x)(\alpha u_1^{'}(x)+\beta u_2^{'}(x))dx + m\cdot\int_a^b(\alpha u_1^{'}(x)+\beta u_2^{'}(x))v(x)dx \\
&\quad + n\cdot\int_a^b(\alpha u_1(x)+\beta u_2(x))v(x)dx \\
&= \alpha(\int_a^b v^{'}(x)u_1^{'}(x)dx + m\cdot\int_a^b u_1^{'}(x)v(x)dx \\
&\quad + n\cdot\int_a^b u_1(x)v(x)dx) \\
&\quad + \beta(\int_a^b v^{'}(x)u_2^{'}(x)dx + m\cdot\int_a^b u_2^{'}(x)v(x)dx \\
&\quad + n\cdot\int_a^b u_2(x)v(x)dx) \\
&= \alpha a(u_1,v)+\beta a(u_2,v)
\end{aligned}
$$

Similarly:

$$
\begin{aligned}
2)a(u,\alpha v_1+\beta v_2) &= \int_a^b(\alpha v_1^{'}(x)+\beta v_2^{'}(x))u^{'}(x)dx + m\cdot\int_a^b u'(x)(\alpha v_1(x)+\beta v_2(x))dx \\
&\quad + n\cdot\int_a^b u(x)(\alpha v_1(x)+\beta v_2(x))dx \\
&= \alpha(\int_a^b v_1^{'}(x)u^{'}(x)dx + m\cdot\int_a^b u^{'}(x)v_1(x)dx \\
&\quad + n\cdot\int_a^b u(x)v_1(x)dx) \\
&\quad + \beta(\int_a^b v_2^{'}(x)u^{'}(x)dx + m\cdot\int_a^b u^{'}(x)v_2(x)dx \\
&\quad + n\cdot\int_a^b u(x)v_2(x)dx) \\
&= \alpha a(u,v_1)+\beta a(u,v_2)
\end{aligned}
$$

$\Rightarrow a(\cdot,\cdot)$ is a bilinear functional

4.5.2 Proof that $\hat{l}(\cdot)$ is linear

Recalling that $\hat{l}(v)=l(v)-a(u_D,v)$

Since it has already been proven that $a(u_D,v)$ is linear in 4.5.1, it needs only be proved that $l(v)$ is linear for $\hat{l}$ to be linear.

$$
\begin{aligned}
l(\alpha v_1+\beta v_2) &= \int_a^b f(x)(\alpha v_1(x)+\beta v_2(x))dx - a(u_0(x),\alpha v_1(x)+\beta v_2(x)) \\
&= \alpha\left(\int_a^b f(x)(v_1(x))dx - a(u_0(x),v_1(x))\right) + \beta\left(\int_a^b f(x)(v_2(x))dx - a(u_0(x),v_2(x))\right)
\end{aligned}
$$

(by the linearity of integrals and the bilinearity of 'a' demonstrated in 4.5.1)

$$= \alpha l(v_1)+\beta l(v_2)$$

$\Rightarrow l(\cdot)$ is a linear functional

$\Rightarrow \hat{l}(\cdot)$ is a linear functional

4.5.3 Proof that $a(\cdot,\cdot)$ is coercive

$a(u,u) = \int_a^b u'(x)^2 + mu'(x)u(x) + nu(x)^2 dx$

The middle term can be evaluated:

$\int_a^b mu'(x)u(x)dx = [mu(x)^2]_a^b - \int_a^b mu(x)u'(x)dx$

Since $u \in H_0^1([a,b])$, the term: $[mu(x)^2]_a^b$ vanishes

$\Rightarrow \int_a^b mu'(x)u(x)dx = -\int_a^b mu'(x)u(x)dx \Rightarrow \int_a^b mu'(x)u(x)dx = 0$

so the equation evaluates to:

$a(u,u) = \int_a^b u'(x)^2 + nu(x)^2 dx$

Then, under the assumption that $n \geq 0$:

$a(u,u) \geq \int_a^b u'(x)^2 dx$

using the Poincaré-Friedrichs inequality (Lemma 1), this can be further extended to:

$a(u,u) \geq \frac{2}{(b-a)^2} \int_a^b u(x)^2 dx$

Therefore, the full result can be derived:

$\left(1 + \frac{(b-a)^2}{2}\right) a(u,u) \geq \int_a^b u(x)^2 + u'(x)^2 dx = ||u(x)||_{H^1}^2$

$\Rightarrow a(u,u) \geq \frac{1}{\left(1+\frac{(b-a)^2}{2}\right)} ||u(x)||_{H^1}^2$

(where n is taken as a non negative value)

4.5.4 Proof that $a(\cdot,\cdot)$ is continuous

$$|a(u,v)| = |\int_a^b v^{'}(x)u^{'}(x)dx + m\int_a^b u^{'}(x)v(x)dx + n\int_a^b u(x)v(x)dx|$$

$$\leq ||v'||_{L^2([a,b])}||u'||_{L^2([a,b])} + m||u'||_{L^2([a,b])}||v||_{L^2([a,b])} + n||u||_{L^2([a,b])}||v||_{L^2([a,b])}$$

(By Cauchy-Schwarz inequality (Lemma 2))

$$\leq \max(1,m,n)\left(||v'||_{L^2([a,b])}||u'||_{L^2([a,b])} + ||u'||_{L^2([a,b])}||v||_{L^2([a,b])} + ||u||_{L^2([a,b])}||v||_{L^2([a,b])}\right)$$

$$\leq c_1\left(||u'||_{L^2([a,b])}^2 + ||u||_{L^2([a,b])}^2\right)^{\frac{1}{2}}\left(||v'||_{L^2([a,b])}^2 + ||v||_{L^2([a,b])}^2\right)^{\frac{1}{2}}$$

$$\leq c_1||u||_{H^1([a,b])}||v||_{H^1([a,b])}$$

4.5.5 Proof that $\hat{l}(\cdot)$ is continuous

Recalling that $\hat{l}(v) = l(v) - a(u_D, v)$

Since it has already been proven that $a(u_D, v)$ is continuous in 4.5.4, it needs only be proved that $l(v)$ is continuous for $\hat{l}$ to be continuous as:

$$|\hat{l}(v)| \leq |l(v)| + |a(u_D, v)|$$

$$\leq |l(v)| + C_1||u_D||_{H^1([a,b])}||v||_{H^1([a,b])}$$

It can then be shown:

$|l(v)| = |\int_a^b f(x)v(x)dx| \leq ||f||_{L^2([a,b])}||v||_{L^2([a,b])}$ (By Cauchy-Schwarz inequality (2))

$$\leq ||f||_{L^2([a,b])}\left(||v||_{L^2([a,b])} + ||v'||_{L^2([a,b])}\right)^{\frac{1}{2}}$$

$$= ||f||_{L^2([a,b])}||v||_{H^1([a,b])}$$

so $\hat{l}(v)$ is continuous.

4.5.6 Summary

Using all the proofs above, it has been shown that, for the construction chosen, there will always be a unique solution to the weak formulation (4.4)

The proof relies on the fact that $n > 0$ so if this is the case, the result can be guaranteed. Experimentation shows that solutions can usually still be found for the case when $n \leq 0$ too, it is just not guaranteed.

If this is satisfied, it will mean that the problem will always be well posed as long as the solution is stable for given inputs.

5 Finite element method

A key component of the finite element method is to replace the infinite dimensional function spaces $H^1_{D(A,B)}$ and H^1_0 with finite dimensional spaces built up as spans of polynomial basis functions.

5.1 Defining basis functions

Denoting the approximation to the true solution as the function $u_h \in V_h \subseteq H^1_{D(A,B)}$, the function is decomposed into a linear composition of basis functions:

$$u_h(x) = \sum_{j=0}^{N} u_j \phi_j(x) \tag{5.1}$$

such that, instead of finding a function to solve the equation, only the coefficients u_j need be found to solve the system.

Similarly to (4.4), this evaluation is split up into:

Definition 5.1. u_h rearrangement

$u_h(x) = u_{h,0} + u_{h,D(A,B)} \in V_h$

where:

$$u_{h,0} = \sum_{j=1}^{N-1} u_j \phi_j(x) \in H^1_0([a,b])$$

$u_{h,D(A,B)} = A\phi_0 + B\phi_N \in H^1_{D(A,B)}([a,b])$

s.t.

$u_h(x) = \sum_{j=1}^{N-1} u_j \phi_j(x) + A\phi_0 + B\phi_N$

such that the relevant vector space is given:

Definition 5.2. V_h definition

$u_h(x) \in V_h = \text{span}\{\phi_1 \dots \phi_{N-1}\} + A\phi_0 + B\phi_N \subseteq H^1_{D(A,B)}$

This rearrangement means that the $u_{h,D(A,B)}$ function can be chosen simply to match the boundary conditions and the $u_{h,0}$ function can be solved by the finite element method which has been shown in section 4 to guarantee a unique solution.

The system being solved for is then given:

Approximation model

Find $u_{h,0}(x) \in V_h$ s.t. $\forall v(x) \in V_h$:

$$a(u_{h,0}, v) = \hat{l}(v)$$
where $a(u,v) = \int_a^b v^{'}(x)u^{'}(x)dx + m \cdot \int_a^b u^{'}(x)v(x)dx + n \cdot \int_a^b u(x)v(x)dx$ (5.2)

and $\hat{l}(v) = l(v) - a(u_{h,D(A,B)}, v)$

for $l(v) = \int_a^b f(x)v(x)dx$

By the linearity of the functional 'a', the system can be rearranged:

$a(u_{h,0}, \phi_i) = a\left(\sum_{j=1}^{N-1} u_j\phi_j(x), \phi_i\right) = \sum_{j=1}^{N-1} u_j a(\phi_j(x), \phi_i)$

Therefore, the system can be rewritten as:

Discretised system

Find $\{u_j\} \in \mathbb{R}^{N-1}$ s.t.

$$\sum_{j=1}^{N-1} u_j a(\phi_j(x), \phi_i(x)) = \hat{l}(\phi_i(x)) \qquad \forall i \in [0, \ldots, N] \tag{5.3}$$

where $a(\phi_j, \phi_i) = \int_a^b \phi_i^{'}(x)\phi_j^{'}(x)dx + m \cdot \int_a^b \phi_j^{'}(x)\phi_i(x)dx + n \cdot \int_a^b \phi_j(x)\phi_i(x)dx$

and $\hat{l}(\phi_i) = \int_a^b f(x)\phi_i(x)dx - A\sum_{j=1}^{N-1} a(\phi_0(x), \phi_j(x)) - B\sum_{j=1}^{N-1} a(\phi_N(x), \phi_j(x))$

This system can then be rewritten as the matrix system:

Matrix system

$\underline{\underline{A}}\underline{U} = \underline{L}$

Where: (5.4)

$A_{i,j} = a(\phi_j, \phi_i) \qquad \forall i, j \in [1 : N-1]$

$U_i = u_i$

$L_i = \hat{l}(\phi_i)$

This is a simple matrix system that can be solved for whatever set of basis functions that are established.

Importantly, if the basis functions are chosen to have a small support (i.e. only be non zero for a small section of the domain), then the matrix will be sparse and easier to solve.

5.2 Defining elements

A key idea behind the finite element method is to break the domain down into a series of intervals or 'elements' on the domain:

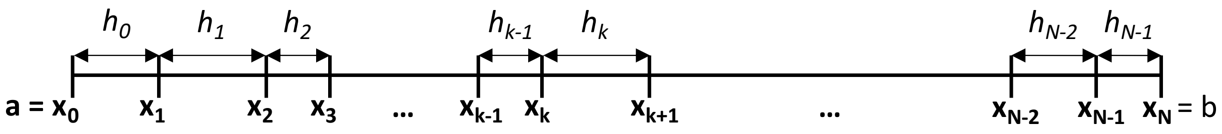

Figure 1: Discretised domain

If the basis functions are polynomials defined largely element-wise then the sparsity mentioned above will be enacted. The structure of this will also allow for local adaptivity which is a key advantage of the finite element method (see section 13).

6 Proving the model yields a sensible approximation

Let the approximation $u_{h,0}(x)$ be posed in the function space $H_0^1([a,b])$ s.t. the weak form is satisfied:

$$a(u_{h,0}, v_h) = \hat{l}(v_h) \qquad \forall v_h \in H_0^1([a,b]) \tag{6.1}$$

Then it is known that the true solution $u(x)$ will further satisfy:

$$a(u_0, v_h) = \hat{l}(v_h) \qquad \forall v_h \in H_0^1([a,b]) \tag{6.2}$$

6.1 Galerkin orthogonality

This is a useful result for later proofs.

6.1.1 Result

$$a(u_0 - u_{h,0}, v_h) = 0 \qquad \forall v_h \in H_0^1([a,b]) \tag{6.3}$$

6.1.2 Proof

Subtracting (6.1) from (6.2) yields:

$a(u_0, v) - a(u_{h,0}, v) = 0$

utilising the bi-linearity of a, these can be combined to give:

$a(u_0 - u_{h,0}, v_h) = 0$

6.2 Céa's Lemma

6.2.1 Result

Céa's lemma is a bound on the error of the approximation. It states that, for a system with the true solution $u(x)$, a finite element approximation $u_h(x) \in V$ has error in the function space norm bounded by:

$$||u - u_h||_V \leq C \min_{v_h \in V} (||u - v_h||_V) \tag{6.4}$$

where in this case, the result desired is:

$$||u_0 - u_{h,0}||_{H^1} \leq C \min_{v_h \in H_0^1([a,b])} (||u_0 - v_h||_{H^1}) \tag{6.5}$$

6.2.2 Proof

By the coercivity result proved in 4.5.3, it is known that:

$$\begin{aligned} &\exists c_0 \in \mathbb{R} \qquad s.t. \\ &c_0||u_0 - u_{h,0}||^2_{H^1} \leq a(u_0 - u_{h,0}, u_0 - u_{h,0}) \end{aligned} \tag{6.6}$$

This can be rearranged in the following way:

$$\begin{aligned} c_0||u_0 - u_{h,0}||^2_{H^1} &\leq a(u_0 - u_{h,0}, u_0 - u_{h,0}) \\ &= a(u_0 - u_{h,0}, u_0) - a(u_0 - u_{h,0}, u_{h,0}) \qquad \text{(by linearity of a)} \\ &= a(u_0 - u_{h,0}, u_0) \qquad \text{(by Galerkin orthogonality (6.3))} \\ &= a(u_0 - u_{h,0}, u_0) - a(u_0 - u_{h,0}, v_h) \qquad \forall v_h \in H_0^1([a,b]) \\ &\qquad \text{(by Galerkin orthogonality (6.3))} \\ &= a(u_0 - u_{h,0}, u_0 - v_h) \qquad \text{(by linearity of a)} \\ &\leq c_1||u_0 - u_{h,0}||_{H^1}||u_0 - v_h||_{H^1} \qquad \text{(for some } c_1 \in \mathbb{R} \text{ by continuity of a (4.5.4))} \end{aligned} \tag{6.7}$$

Dividing through by $||u_0 - u_{h,0}||_{H^1}$ and rearranging yields:

$$\begin{aligned} &\exists c_1, c_0 \in \mathbb{R} \text{ s.t. } \forall v_h \in H_0^1([a,b]) : \\ &||u_0 - u_{h,0}||_{H^1} \leq \frac{c_1}{c_0}||u_0 - u_{h,0}||_{H^1} \leq C \min_{v_h \in H_0^1([a,b])} (||u_0 - v_h||_{H^1}) \\ &\text{(For } C = \frac{c_1}{c_0}\text{)} \end{aligned} \tag{6.8}$$

6.2.3 Summary

Céa's lemma tells us that the approximation $u_h(x)$ being constructed in some function space is a near-best fit to u_0 in the parent function space the problem is set up in.

This means that the method used is producing the best possible estimate to the solution in the function space being used (up to some constant C).

Specifically, in the setup established, the result is given:

$$||u_0 - u_{h,0}||_{H^1} \leq C_1 \min_{v_h \in H_0^1([a,b])} (||u_0 - v_h||_{H^1}) \tag{6.9}$$

Given $||u_0 - u_{h,0}||_{L^2} \leq ||u_0 - u_{h,0}||_{H^1}$ by its definition, it can also be concluded:

$$||u_0 - u_{h,0}||_{L^2} \leq C_2 \min_{v_h \in H_0^1([a,b])} (||u_0 - v_h||_{H^1}) \tag{6.10}$$

7 Applying a first order finite element method

7.1 First order model

7.1.1 Choice of basis

For a linear approximation, the bases are set up as transformations of the following simple functions on each element:

$$\hat{\phi}_{\text{down}}(\eta) = \begin{cases} 1-\eta & \text{if } 0 \leq \eta \leq 1, \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

$$\hat{\phi}_{\text{up}}(\eta) = \begin{cases} \eta & \text{if } 0 \leq \eta \leq 1, \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

These can be easily transferred to each element using the following prescription:

$$\phi_k(x) = \begin{cases} \hat{\phi}_{\text{up}}(\frac{x-x_{k-1}}{h_{k-1}}) = \frac{x-x_{k-1}}{h_{k-1}} & \text{if } x_{k-1} \leq x \leq x_k, & 1 \leq k \leq N \\ \hat{\phi}_{\text{down}}(\frac{x-x_k}{h_k}) = \frac{x_{k+1}-x}{h_k} & \text{if } x_k \leq x \leq x_{k+1}, & 0 \leq k \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{7.3}$$

This gives the whole domain a structure like:

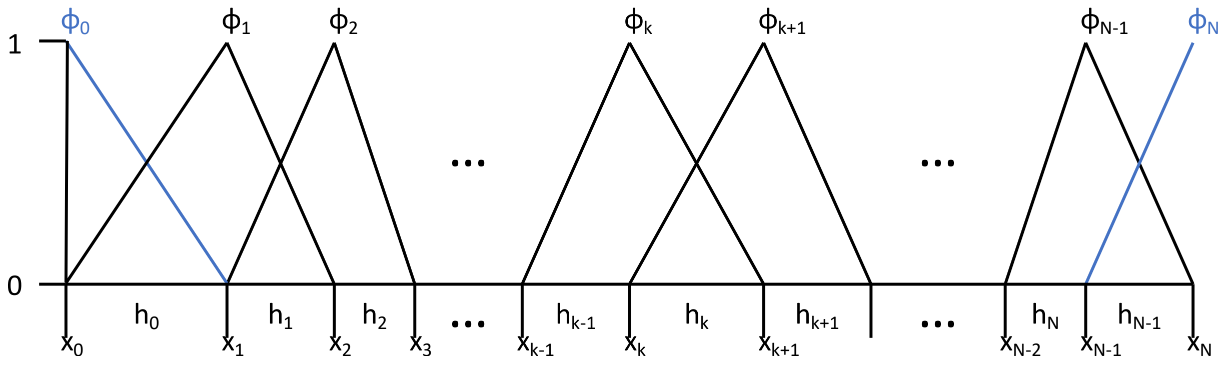

Figure 2: Linear basis full domain

Which will be used to generate the piecewise linear function $u_h(x) = u_{h,0}(x) + u_{h,D}(x)$ as prescribed in (5.1) where:

$u_{h,0}(x) = \sum_{k=1}^{N-1} u_k \phi_k(x)$

$u_{h,D}(x) = A\phi_0(x) + B\phi_N(x)$

7.1.2 Proof linear model is consistent with established spaces

The model proposed seems like a sensible choice but it is important to check that it is consistent with the space set up in (5.2). i.e. $u_h(x) \in V_h$ by showing $u_{h,0}(x) \in H_0^1([a,b])$ and $u_{h,D}(x) \in H^1_{D(A,B)}([a,b])$.

This can be done by showing that $u_h \in H^1([a,b])$ as well as $u_{h,0}(a) = u_{h,0}(b) = 0$ and $u_{h,0}(a) = A$ and $u_{h,0}(b) = B$ with the latter being trivially true from their setup.

Using the definition provided in (2.7), the remaining proof can be achieved by proving $u_h \in L^2([a,b])$ and $D_1(u_h) \in L^2([a,b])$ where $D_1(u_h)$ is the weak derivative of the function established.

<u>1) Show $u_h \in L^2([a,b])$</u>

Recalling the definition in (2.4), this result is equivalent to proving:

Show $||u_h||_{L^2([a,b])} = \left(\int_a^b |u_h(x)|^2 dx\right)^{\frac{1}{2}} < \infty$

Applying the Cauchy-Schwarz inequality (Lemma 2) to $\mathbb{R}^N$ gives the following result:

$$\sum_{i=1}^{N} x_i y_i \leq \left(\sum_{i=1}^{N} x_i^2\right)^{\frac{1}{2}} \left(\sum_{i=1}^{N} y_i^2\right)^{\frac{1}{2}} \tag{7.4}$$

Using this result, the following result can be stated:

$$|u_h(x)|^2 = |\sum_{k=1}^{N-1} u_k\phi_k(x) + A\phi_0(x) + B\phi_N(x)|^2$$
$$\leq \sum_{k=1}^{N-1} |u_i|^2|\phi_k(x)|^2 + |A|^2|\phi_0(x)|^2 + |B|^2|\phi_N(x)|^2$$

Since $\forall k \in [0:N], \phi_k(x)$ is continuous on the domain

$\exists M_k < \infty \quad s.t. \quad |\phi_k(x)| < M_k \quad \forall x \in [a,b]$

$\Rightarrow ||u_h||_{L^2([a,b])} \leq \left(\sum_{k=0}^{N} \left(\int_{x_{k-1}}^{x_{k+1}} |u_k|^2|\phi_k(x)|^2 dx\right)\right)^{\frac{1}{2}} < \sum_{k=0}^{N} |u_k|^2 M_k^2 (x_{k+1} - x_{k-1}) < \infty$

$\Rightarrow u_h \in L^2([a,b])$

<u>2) Calculate $D_1(\phi_k(x)) \forall k \in [1:N-1]$</u>

Recalling the definition in 2.5, this means:

Find $D_1(\phi_k(x))$ s.t. $\int_a^b \phi_k(x)v'(x)dx = -\int_a^b D_1(\phi_k(x))v(x)dx$

Starting from the left hand side, the following manipulations can be made:

$$\begin{aligned}
\int_a^b \phi_k(x)v'(x)dx &= \int_{x_{k-1}}^{x_{k+1}} \phi_k(x)v'(x)dx \\
&= \int_{x_k}^{x_{k+1}} \phi_k(x)v'(x)dx + \int_{x_{k-1}}^{x_k} \phi_k(x)v'(x)dx \\
&= [\phi_k(x)v(x)]_{x_{k-1}}^{x_k} - \int_{x_{k-1}}^{x_k} \phi_k'(x)v(x)dx + [\phi_k(x)v(x)]_{x_k}^{x_{k+1}} - \int_{x_k}^{x_{k+1}} \phi_k'(x)v(x)dx \\
&= \left[\frac{x - x_{k-1}}{h_{k-1}}v(x)\right]_{x_{k-1}}^{x_k} - \int_{x_{k-1}}^{x_k} \frac{1}{h_{k-1}}v(x)dx + \left[\frac{x_{k+1} - x}{h_k}v(x)\right]_{x_k}^{x_{k+1}} - \int_{x_k}^{x_{k+1}} \frac{1}{h_k}v(x)dx \\
&= v(x_k) - \int_{x_{k-1}}^{x_k} \frac{1}{h_{k-1}}v(x)dx - v(x_k) - \int_{x_k}^{x_{k+1}} \frac{1}{h_k}v(x)dx \\
&= -\left(\int_{x_{k-1}}^{x_k} \frac{1}{h_{k-1}}v(x)dx + \int_{x_k}^{x_{k+1}} \frac{1}{h_k}v(x)dx\right)
\end{aligned}$$

By inspection, it can clearly be seen that the derivative is therefore given by:

$$D_1(\phi_k(x)) = \begin{cases} \frac{1}{h_{k-1}} & x_{k-1} < x < x_k \\ \frac{-1}{h_k} & x_k < x < x_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

This is an important result as these ϕ_k functions don't have a classical derivative due to the jumps at the piecewise boundaries and yet the weak derivatives exist and are well defined.

Note also that the values at x_k,x_{k-1} and x_{k+1} could actually be defined as any constant as infinitesimal points have no impact on integrals, the above prescription is just a simple example of a weak derivative that fulfils the relevant conditions.

3) Calculate $D_1(\phi_0(x))$ and $D_1(\phi_N(x))$

In a similar derivation to the previous functions, it can easily be determined that these weak derivatives also exist and are given:

$$D_1(\phi_0(x)) = \begin{cases} \frac{-1}{h_0} & x_0 < x < x_1 \\ 0 & \text{otherwise} \end{cases} \tag{7.6}$$

and

$$D_1(\phi_N(x)) = \begin{cases} \frac{1}{h_{N-1}} & x_{N-1} < x < x_N \\ 0 & \text{otherwise} \end{cases} \tag{7.7}$$

4) Show $D_1(u_h(x)) \in L^2([a,b])$

By (7.5), it can clearly be seen that:

$D_1(u_h(x)) = \sum_{k=1}^{N-1} u_k D_1(\phi_k(x)) + A \cdot D_1(\phi_0(x)) + B \cdot D_1(\phi_N(x))$

is actually just a constant and therefore is trivially a member of $L^2([a,b])$

Summary

By the arguments above, it has been proved that $u_h(x) \in V_h$

Therefore the results established in this function space hold for this model . Particularly, by the proof in section 4.3, there exists a unique solution when the problem is set up this way.

7.2 First order linear system

7.2.1 Computing the linear system element-wise

In order to solve the matrix system established in (5.4), several evaluations of the functionals $a(\phi_j, \phi_i) = \int_a^b \phi_i'(x)\phi_j'(x)dx + m \cdot \int_a^b \phi_j'(x)\phi_i(x)dx + n \cdot \int_a^b \phi_j(x)\phi_i(x)dx$ and $\hat{l}(\phi_i) = \int_a^b f(x)\phi_i(x)dx - A\sum_{j=1}^{N-1} a(\phi_0(x), \phi_j(x)) - B\sum_{j=1}^{N-1} a(\phi_N(x), \phi_j(x))$ need to be calculated.

These evaluations can be built up elementwise as, by the additivity of integrals, each integral in the matrix is simply the sum of the integrals over all elements.

Each element for $k \in [1 : N-2]$ looks like:

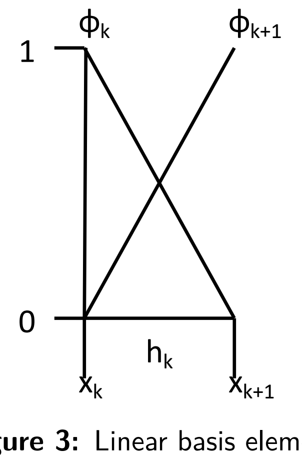

Figure 3: Linear basis element

with the end two intervals just missing one of these functions respectively.

It is therefore clear that each element $[x_k, x_k + 1]$ can only give a contribution to the integrals $a(\phi_j, \phi_i)$ and $\hat{l}(\phi_i)$ for $i, j \in [k, k+1]$ as all other combinations will have a zero factor and therefore evaluate as zeroes.

This demonstrates the power of a small support as this will result in a sparse (tridiagonal) matrix which can be built up iteratively using:

$$\begin{aligned} a(\phi_j, \phi_i) &= \int_a^b \phi_i^{'}(x)\phi_j^{'}(x) + m\phi_j^{'}(x)\phi_i(x)dx + n\phi_j(x)\phi_i(x)dx \\ &= \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} \phi_i^{'}(x)\phi_j^{'}(x) + m\phi_j^{'}(x)\phi_i(x)dx + n\phi_j(x)\phi_i(x)dx \qquad \text{for } i, j \in [k, k+1] \end{aligned} \tag{7.8}$$

7.2.2 Change of coordinates

By having all the basis functions be transformations of the same shape, calculations can be further simplified by the following process:

On the interval $[x_k, x_{k+1}]$, the functions are given as in (7.3):

$\phi_k(x) = \hat{\phi}_{\text{down}}(\frac{x-x_k}{h_k})$

$\phi_{k+1}(x) = \hat{\phi}_{\text{up}}(\frac{x-x_k}{h_k})$

So, letting $\eta = \frac{x-x_k}{h_k} \in [0, 1]$, change of variables gives:

$$\begin{aligned} \int_{x_k}^{x_{k+1}} \phi_k(x)dx &= \int_{x_k}^{x_{k+1}} \hat{\phi}_{\text{down}}(\frac{x-x_k}{h_k})dx \\ &= \int_0^1 \hat{\phi}_{\text{down}}(\frac{(h_k\eta + x_k) - x_k}{h_k})\frac{dx}{d\eta}d\eta \\ &= h_k \int_0^1 \hat{\phi}_{\text{down}}(\eta)d\eta \\ \int_{x_k}^{x_{k+1}} \phi_{k+1}(x)dx &= \int_{x_k}^{x_{k+1}} \hat{\phi}_{\text{up}}(\frac{x-x_k}{h_k})dx \\ &= \int_0^1 \hat{\phi}_{\text{up}}(\frac{(h_k\eta + x_k) - x_k}{h_k})\frac{dx}{d\eta}d\eta \\ &= h_k \int_0^1 \hat{\phi}_{\text{up}}(\eta)d\eta \end{aligned} \tag{7.9}$$

Similarly:

$$
\begin{aligned}
\int_{x_k}^{x_{k+1}} \phi'_k(x)dx &= \int_{x_k}^{x_{k+1}} \frac{d}{dx}\left(\hat{\phi}_{\textsf{down}}(\frac{x-x_k}{h_k})\right)dx \\
&= \frac{1}{h_k}\int_{x_k}^{x_{k+1}} \hat{\phi}'_{\textsf{down}}(\frac{x-x_k}{h_k})dx \\
&= \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\textsf{down}}(\frac{(h_k\eta + x_k) - x_k}{h_k})\frac{dx}{d\eta}d\eta \\
&= \int_0^1 \hat{\phi}'_{\textsf{down}}(\eta)d\eta \\
\int_{x_k}^{x_{k+1}} \phi'_{k+1}(x)dx &= \int_{x_k}^{x_{k+1}} \frac{d}{dx}\left(\hat{\phi}_{\textsf{up}}(\frac{x-x_k}{h_k})\right)dx \\
&= \frac{1}{h_k}\int_{x_k}^{x_{k+1}} \hat{\phi}'_{\textsf{up}}(\frac{x-x_k}{h_k})dx \\
&= \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\textsf{up}}(\frac{(h_k\eta + x_k) - x_k}{h_k})\frac{dx}{d\eta}d\eta \\
&= \int_0^1 \hat{\phi}'_{\textsf{up}}(\eta)d\eta
\end{aligned}
\tag{7.10}
$$

This means that the only integrals which need to be calculated are the integrals of the core functions $\hat{\phi}_{\textsf{up}}$ and $\hat{\phi}_{\textsf{down}}$ and their derivatives.

This means that these calculations only need to be done once instead of for every combination of functions for every element. This makes computation simpler and the code more efficient.

7.2.3 Explicitly evaluating the linear system using change of coordinates

Using a similar method to the ones shown in (7.9) and (7.10), the non zero integrals on a general interval can be decomposed:

$$
\begin{aligned}
\int_{x_k}^{x_{k+1}} \phi'_k(x)\phi'_k(x)dx &= \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\textsf{down}}(\eta)\hat{\phi}'_{\textsf{down}}(\eta)d\eta = \frac{1}{h_k} \\
\int_{x_k}^{x_{k+1}} \phi'_k(x)\phi'_{k+1}(x)dx &= \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\textsf{down}}(\eta)\hat{\phi}'_{\textsf{up}}(\eta)d\eta = \frac{-1}{h_k} \\
\int_{x_k}^{x_{k+1}} \phi'_{k+1}(x)\phi'_k(x)dx &= \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\textsf{up}}(\eta)\hat{\phi}'_{\textsf{down}}(\eta)d\eta = \frac{-1}{h_k} \\
\int_{x_k}^{x_{k+1}} \phi'_{k+1}(x)\phi'_{k+1}(x)dx &= \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\textsf{down}}(\eta)\hat{\phi}'_{\textsf{down}}(\eta)d\eta = \frac{1}{h_k}
\end{aligned}
\tag{7.11}
$$

$$\begin{aligned}
\int_{x_k}^{x_{k+1}} \phi_k'(x)\phi_k(x)dx &= \int_0^1 \hat{\phi}_{\text{down}}'(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta = \frac{-1}{2}\\
\int_{x_k}^{x_{k+1}} \phi_k'(x)\phi_{k+1}(x)dx &= \int_0^1 \hat{\phi}_{\text{down}}'(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta = \frac{-1}{2}\\
\int_{x_k}^{x_{k+1}} \phi_{k+1}'(x)\phi_k(x)dx &= \int_0^1 \hat{\phi}_{\text{up}}'(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta = \frac{1}{2}\\
\int_{x_k}^{x_{k+1}} \phi_{k+1}'(x)\phi_{k+1}(x)dx &= \int_0^1 \hat{\phi}_{\text{up}}'(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta = \frac{1}{2}
\end{aligned} \tag{7.12}$$

and

$$\begin{aligned}
\int_{x_k}^{x_{k+1}} \phi_k(x)\phi_k(x)dx &= h_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta = \frac{h_k}{3}\\
\int_{x_k}^{x_{k+1}} \phi_k(x)\phi_{k+1}(x)dx &= h_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta = \frac{-1}{6h_k}\\
\int_{x_k}^{x_{k+1}} \phi_{k+1}(x)\phi_k(x)dx &= h_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta = \frac{-h_k}{6}\\
\int_{x_k}^{x_{k+1}} \phi_{k+1}(x)\phi_{k+1}(x)dx &= h_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta = \frac{h_k}{3}
\end{aligned} \tag{7.13}$$

7.2.4 Initial pseudo-algorithm

Remembering that the elements of the linear system were given in (5.3) and (5.4) by:

$\forall i, j \in [1 : N-1]$:

$A_{i,j} = a(\phi_j, \phi_i) = \int_a^b \phi_i'(x)\phi_j'(x)dx + m \cdot \int_a^b \phi_j'(x)\phi_i(x)dx + n \cdot \int_a^b \phi_j(x)\phi_i(x)dx$

and

$L_i = \hat{l}(\phi_i) = \int_a^b f(x)\phi_i(x)dx - A \cdot a(\phi_0, \phi_i) - B \cdot a(\phi_N, \phi_i)$

Utilising the element-wise evaluation of (7.8) where all integrals are just rescalings of a small system of precomputed integrals of base functions as in section 7.2.3, the system can be resolved.

Importantly, here, in order to resolve all values at the same time, u_0 and u_N are included in trivial equations to be resolved as $u_0 = A$ and $u_N = B$ and therefore resolve the full model. This doesn't need to be done as the values are already known but it just means all values are resolved with one system in one place which makes the code neater.

Utilising the methods highlighted, the full algorithm can be demonstrated as:

```
1: procedure Full linear system setup
2:
        $A = L = 0$
        $\underline{u(a) = A}$
3:      $A_{0,0} = 1$
        $L_0 = A$
4:      for all $k \in [0 : N-1]$ do
5:          if $k \neq 0$ and $k \neq N-1$ then
                $\underline{A_{k+1,k}}$
                $A_{k+1,k} = A_{k+1,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
                $\qquad +nh_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
                $\underline{A_{k,k+1}}$
                $A_{k,k+1} = A_{k,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
                $\qquad +nh_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
6:          else if $k \neq 0$ then
                $\underline{A_{k,k}}$
                $A_{k,k} = A_{k,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
                $\qquad +nh_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
7:          else if $k \neq N-1$ then
                $\underline{A_{k+1,k+1}}$
                $A_{k+1,k+1} = A_{k+1,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
                $\qquad +nh_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
8:          end if
9:          if $k \neq 0$ then
                $\underline{L_k}$
                $L_k = L_k + \int_{x_{k-1}}^{x_{k+1}} f(x)\phi_k(x)dx$
10:         end if
11:     end for
```

```
        $\underline{u(b) = B}$
12:     $A_{N,N} = 1$
        $L_N = B$
        Boundary corrections
13:     $L_1 = L_1 - A\left(\frac{1}{h_0}\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}'_{\mathsf{down}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta + nh_0\int_0^1 \hat{\phi}_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta\right)$
14:     $L_{N-1} = L_{N-1} - B\left(\frac{1}{h_{N-1}}\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}'_{\mathsf{up}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta + nh_{N-1}\int_0^1 \hat{\phi}_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta\right)$
15:     return $A, L$
16: end procedure
```

Which in this case evaluates the system explicitly as:

System with the boundary corrections

$$\begin{bmatrix} 1 & & & & \\ & \begin{pmatrix} (\frac{1}{h_0} - \frac{m}{2} + \frac{nh_0}{3}) \\ +(\frac{1}{h_1} + \frac{m}{2} + \frac{nh_1}{3}) \end{pmatrix} & \begin{pmatrix} \frac{-1}{h_1} + \frac{m}{2} \\ - \frac{nh_1}{6} \end{pmatrix} & & \\ & \begin{pmatrix} \frac{-1}{h_1} - \frac{m}{2} \\ - \frac{nh_1}{6} \end{pmatrix} & \ddots & & \\ & & \ddots & & \\ & & \ddots & \begin{pmatrix} \frac{-1}{h_{N-2}} + \frac{m}{2} \\ - \frac{nh_{N-2}}{6} \end{pmatrix} & \\ & & \begin{pmatrix} \frac{-1}{h_{N-2}} - \frac{m}{2} \\ - \frac{nh_{N-2}}{6} \end{pmatrix} & \begin{pmatrix} (\frac{1}{h_{N-2}} - \frac{m}{2} + \frac{nh_{N-2}}{3}) \\ +(\frac{1}{h_{N-1}} + \frac{m}{2} + \frac{nh_{N-1}}{3}) \end{pmatrix} & \\ & & & & 1 \end{bmatrix}$$

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} A \\ \int_{x_0}^{x_2} f(x)\phi_1(x)dx - \left(\frac{-1}{h_0} - \frac{m}{2} - \frac{nh_0}{6}\right) \cdot A \\ \int_{x_1}^{x_3} f(x)\phi_2(x)dx \\ \vdots \\ \int_{x_{N-2}}^{x_{N-1}} f(x)\phi_{N-2}(x)dx \\ \int_{x_{N-1}}^{x_N} f(x)\phi_{N-1}(x)dx - \left(\frac{-1}{h_{N-1}} + \frac{m}{2} - \frac{nh_{N-1}}{6}\right) \cdot B \\ B \end{bmatrix} \tag{7.14}$$

7.3 First order approximation numerical example

Using the method outlined, code was written to generate and solve the linear system for a given input (see Appendix A)

This was then applied to the following problem:

Model problem

$$-u''(x) + mu'(x) + nu(x) = 2k^2\text{tanh}(kx)\text{sech}^2(kx) + m \cdot k\text{sech}^2(kx) + n \cdot \text{tanh}(kx)$$

For $x \in [a, b]$ where $u(a) = \text{tanh}(k \cdot a), u(b) = \text{tanh}(k \cdot b)$

For $a = -0.05, b = 0.05, k = 100, m = 10, n = -20$

(7.15)

This was chosen such that the solution is given by $u(x) = tanh(kx)$. This was chosen as it is essentially a continuous version of the step function where a larger k value makes the function look like a steeper step. This non-smooth behaviour would be difficult for another method to approximate effectively so this function should highlight the efficacy of the finite element method, particularly when it comes to the adaptivity methods.

Below is the linear approximation function evaluated at 5 evenly spaced intervals.

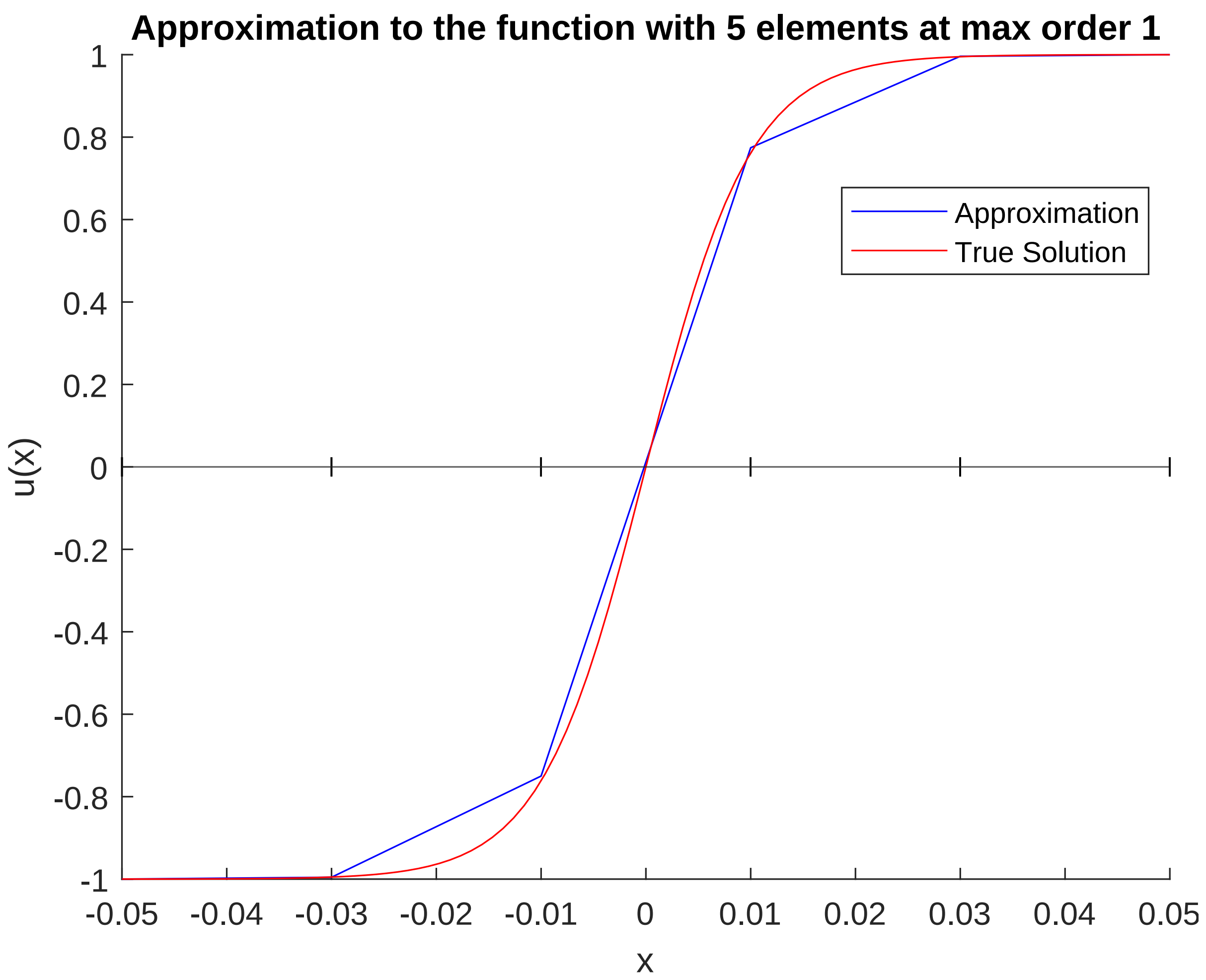

Figure 4: Linear approximation in action

Clearly, just by inspection, the model seems to be approximating the solution correctly with a piecewise continuous linear approximation matching the true solution pretty much exactly at all nodal points given.

8 Implementing a second order approximation

Now that the linear system has been built and verified, it stands to reason that the next step would be to increase the order of approximation.

This is done hierarchically i.e. the basis functions established for the linear system are maintained and new basis functions are just introduced to account for higher order behaviour.

8.1 Second order model

8.1.1 Quadratic basis

As before, the quadratic bases are set up as simple transformations of a single shape. In this case, the core function is simply given as:

$$\hat{\phi}_{\text{quad}}(\eta) = \begin{cases} -\eta(\eta-1) & \text{if } 0 \le \eta \le 1, \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

So all elements can be calculated as:

$$\phi_{k+N}(x) = \begin{cases} \hat{\phi}_{\text{quad}}\left(\frac{x-x_k}{h_k}\right) & \text{if } x_k \le x \le x_{k+1}, \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

making the full domain looking like:

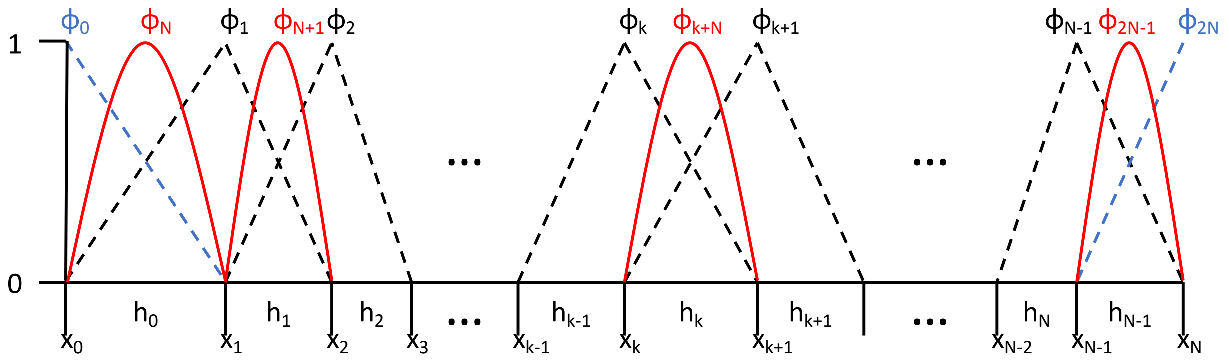

Figure 5: Quadratic basis full domain

8.1.2 Second order approximation model

All this changes about the formulation is that there are a few more basis functions with corresponding coefficients:

$$\begin{aligned} u_h(x) &= \sum_{k=1}^{2N-1} u_k\phi_k(x) + A\phi_0(x) + B\phi_{2N}(x) \\ &= u_0^1(x) + u_0^2(x) + u_{D(A,B)}^1(x) \end{aligned} \tag{8.3}$$

where u^p denotes the contributions at pth order.

Note that the final upslope in $u_{D(A,B)}^1$ moved to the final coefficient u_{2N} to separate out the boundary conditions from the internal structure in the linear system.

8.1.3 Confirming space consistency

In order to be a consistent model and to use the results found based on the space currently being used, it needs to be shown that $u_h(x) \in V_h^2 \subseteq H_{D(A,B)}^1([a,b])$ as before where the function space has been slightly modified to:

$V_h^2 = span\{\phi_1, \cdots \phi_{2N-1}\} + A\phi_0 + B\phi_{2N}$

Since it was already proved in section 7.1.2 that $u_{h,0}(x) = \sum_{k=1}^{N-1} u_k\phi_k(x) \in H_0^1([a,b])$ and $u_{h,D}(x) = A\phi_0 + B\phi_{2N} \in H_{D(A,B)}^1$, it only remains to be proved that $u_{h,0}^2(x) = \sum_{k=0}^{N-1} u_{k+N}\phi_{k+N}(x) \in H_0^1([a,b])$ to conclude $u_h(x) \in V_h^2 \subseteq H_{D(A,B)}^1([a,b])$

This is quite simple to do in the following steps:

1) Show $u_{h,0}^2(x) \in L^2([a,b])$

Recalling the definition in (2.4), this result is equivalent to proving:

Show $||u_{h,0}^2||_{L^2([a,b])} = \left(\int_a^b |u_{h,0}^2(x)|^2 dx\right)^{\frac{1}{2}} < \infty$

Due to the fact that these bases don't overlap, this can simplified to:

$\left(\int_a^b |u_{h,0}^2(x)|^2 dx\right)^{\frac{1}{2}} = \left(\sum_{k=0}^{N-1} |u_{k+N}|^2 \int_{x_k}^{x_{k+1}} \phi_{k+N}(x)dx\right)^{\frac{1}{2}}$

Since all ϕ_{k+N} are continuous on the interval, these integrals will all be finite so:

$||u_{h,0}^2||_{L^2([a,b])} = \left(\sum_{k=0}^{N-1} |u_{k+N}|^2 \int_{x_k}^{x_{k+1}} \phi_{k+N}(x)dx\right)^{\frac{1}{2}} < \infty$

2) Calculate $D_1(\phi_{k+N}(x))$

Recalling the definition in (2.5), this means:

Find $D_1(\phi_{k+N}(x))$ s.t. $\int_a^b \phi_{k+N}(x)v'(x)dx = -\int_a^b D_1(\phi_{k+N}(x))v(x)dx$

Starting from the left hand side, the following manipulations can be made:

$$\begin{aligned}\int_a^b \phi_{k+N}(x)v'(x)dx &= \int_{x_k}^{x_{k+1}} \phi_{k+N}(x)v'(x)dx \\ &= [\phi_{k+N}(x)v(x)]_{x_{k-1}}^{x_k} - \int_{x_{k-1}}^{x_k} \phi'_{k+N}(x)v(x)dx \\ &= -\int_{x_{k-1}}^{x_k} \phi'_{k+N}(x)v(x)dx\end{aligned}$$

By inspection, it can clearly be seen that the derivative is therefore given by the point-wise derivative of the function:

$$D_1(\phi_{k+N}(x)) = \begin{cases} \phi'_{k+N}(x) = \frac{-1}{h_k}\left(2\left(\frac{x-x_k}{h_k}\right)-1\right) & x_k < x < x_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{8.4}$$

3) Show $D_1(u^2_{h,0}(x)) \in L^2([a,b])$

By (8.4), it can clearly be seen that:

$D_1(u^2_{h,0}(x)) = \sum_{k=0}^{N-1} u_k D_1(\phi_{k+N}(x))$

has a piecewise linear representation on the mesh. This is clearly a member of $L^2([a,b])$ as, similarly to section 7.1.2, it can be shown that the L^2 norm of the function is just a finite sum of finite integrals given the function is continuous on the intervals.

4) Show $u^2_{h,0}(a) = u^2_{h,0}(b) = 0$

This is trivial as this is a condition of the basis setup here.

Summary

By the arguments above, it has been proved that $u^2_{h,0}(x) = \sum_{k=0}^{N-1} u_{k+N}\phi_{k+N}(x) \in H^1_0([a,b])$ and, since section 7.1.2 already confirmed $u_{h,0}(x) = \sum_{k=0}^{N} u_k\phi_k(x) \in H^1_0([a,b])$ and $u_{h,D}(x) = A\phi_0 + B\phi_{2N} \in H^1_{D(A,B)}$, by the linearity of vector spaces:

$u_h(x) = \sum_{k=1}^{2N-1} u_k\phi_k(x) + A\phi_0(x) + B\phi_{2N}(x) \in V_h \subseteq H^1_{D(A,B)}([a,b])$

Therefore the new basis is consistent with the previous basis and the results established for both.

8.2 Second order linear system

8.2.1 Second order linear system prescription

By the setup in (8.3), the finite element method system is now a $(2N-1)\times(2N-1)$ system instead of the $(N-1)\times(N-1)$ system in 7.14. The setup follows exactly same as the general setup in (5.3) and (5.4):

Find $\{u_j\} \in \mathbb{R}^{2N-1}$ s.t.

$$\sum_{j=1}^{2N-1} u_j a(\phi_j(x), \phi_i(x)) = l(\phi_i(x)) - A \cdot a(\phi_0(x), \phi_i(x)) - B \cdot a(\phi_{2N}(x), \phi_i(x)) \qquad \forall i \in [1, \dots, 2N-1] \tag{8.5}$$

Which rearranges to the matrix system:

$\underline{\underline{A}}\underline{U} = \underline{L}$

Where,

$$\begin{aligned} &\forall i, j \in [1 : 2N-1] : \\ &A_{i,j} = a(\phi_j, \phi_i) = \int_a^b \phi_i'(x)\phi_j'(x)dx + m \cdot \int_a^b \phi_j'(x)\phi_i(x)dx + n \cdot \int_a^b \phi_j(x)\phi_i(x)dx \\ &U_i = u_i \\ &L_i = \hat{l}(\phi_i) = \int_a^b f(x)\phi_i(x)dx - \sum_{i=0}^{2N} (A \cdot a(\phi_0(x), \phi_i(x)) + B \cdot a(\phi_{2N}(x), \phi_i(x))) \end{aligned} \tag{8.6}$$

All that changes is that there are a few more basis functions and therefore more non zero elements of the matrix.

8.2.2 Setting up the second order linear system

Now, since a generic element looks like:

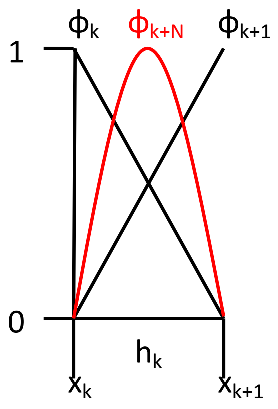

Figure 6: Quadratic basis element

It can be clearly seen that, iterating element-wise as in (7.8), each element $[x_k, x_{k+1}]$ will provide non zero contributions to the matrix at elements $A_{k,k}$, $A_{k,k+1}$, $A_{k,k+N}$, $A_{k+1,k}$, $A_{k+N,k}$ and $A_{k+N,k+N}$

Utilising the same change of coordinates method demonstrated in (7.9), these contributions are simply rescalings of the base shape's interactions with the other base shapes and their derivatives so these can be stored as before to make the integral calculations easier.

Utilising the same method as demonstrated in algorithm 1 adding in the new terms incurred by the quadratic function interactions in the integrals gives the following algorithm:

```
1: procedure FULL LINEAR SYSTEM SETUP QUADRATIC
       $\underline{u(a) = A}$
2:     $A_{0,0} = 1$
       $L_0 = A$
3:     for all $k \in [0 : N-1]$ do
4:         if $k \neq 0$ and $k \neq N-1$ then
               $\underline{A_{k+1,k}}$
               $A_{k+1,k} = A_{k+1,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
                      $+nh_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
               $\underline{A_{k,k+1}}$
               $A_{k,k+1} = A_{k,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
                      $+nh_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
5:         else if $k \neq 0$ then
```

```
            $\underline{A_{k,k}}$
            $A_{k,k} = A_{k,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}'_{\mathsf{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta$
            $\underline{A_{k+N,k}}$
            $A_{k+N,k} = A_{k+N,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{quad}}(\eta)\hat{\phi}'_{\mathsf{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{quad}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{quad}}(\eta)d\eta$
            $\underline{A_{k,k+N}}$
            $A_{k,k+N} = A_{k,k+N} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}'_{\mathsf{quad}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{quad}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{quad}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta$
6:      else if $k \neq N-1$ then
            $\underline{A_{k+1,k+1}}$
            $A_{k+1,k+1} = A_{k+1,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}'_{\mathsf{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta$
            $\underline{A_{k+N,k+1}}$
            $A_{k+N,k+1} = A_{k+N,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{quad}}(\eta)\hat{\phi}'_{\mathsf{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{quad}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{quad}}(\eta)d\eta$
            $\underline{A_{k+1,k+N}}$
            $A_{k+1,k+N} = A_{k+1,k+N} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}'_{\mathsf{quad}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{quad}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{quad}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta$
7:      else
            $\underline{A_{k+N,k+N}}$
            $A_{k+N,k+N} = A_{k+N,k+N} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{quad}}(\eta)\hat{\phi}'_{\mathsf{quad}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\mathsf{quad}}(\eta)\hat{\phi}_{\mathsf{quad}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\mathsf{quad}}(\eta)\hat{\phi}_{\mathsf{quad}}(\eta)d\eta$
8:      end if
9:      if $k \neq 0$ then
            $\underline{L_k}$
            $L_k = L_k + \int_{x_{k-1}}^{x_{k+1}} f(x)\phi_k(x)dx$
```

```
10:         end if
11:     end for
        $\underline{u(b) = B}$
12:     $A_{N,N} = 1$
        $L_N = B$
        Boundary corrections
13:     $L_1 = L_1 - A\left(\frac{1}{h_0}\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta + nh_0\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta\right)$
14:     $L_N = L_N - A\left(\frac{1}{h_0}\int_0^1 \hat{\phi}'_{\text{quad}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{quad}}(\eta)d\eta + nh_0\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{quad}}(\eta)d\eta\right)$
15:     $L_{N-1} = L_{N-1} - B\left(\frac{1}{h_{N-1}}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta + nh_{N-1}\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta\right)$
16:     $L_{2N-1} = L_{2N-1} - B\left(\frac{1}{h_{N-1}}\int_0^1 \hat{\phi}'_{\text{quad}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{quad}}(\eta)d\eta + nh_{N-1}\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{quad}}(\eta)d\eta\right)$
17:     return $A, L$
18: end procedure
```

Applied properly, this should give a linear system with sparsity pattern looking like:

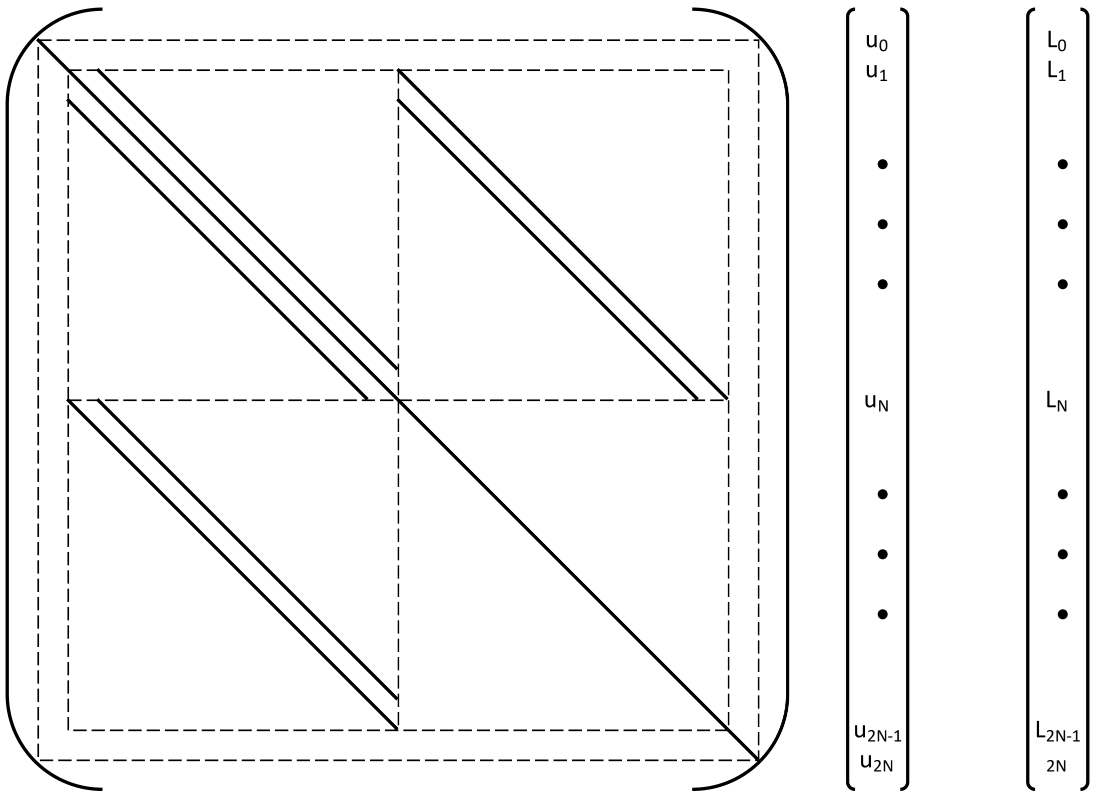

Figure 7: Quadratic system sparsity pattern

8.3 Second order approximation numerical example

Solving the system set up above and substituting back into the model yields the following approximation to the same problem given in (7.15)

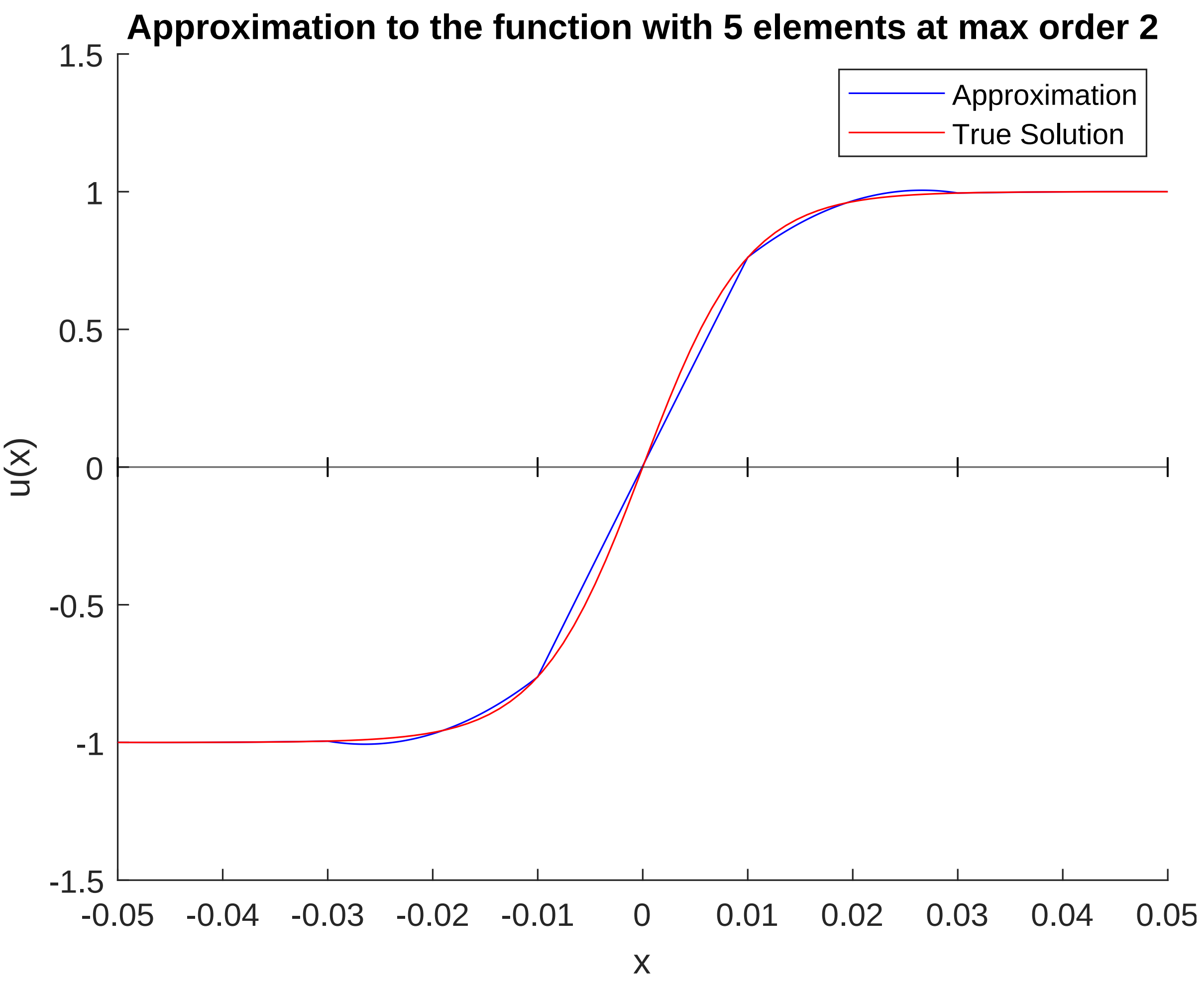

Figure 8: Quadratic approximation in action

This is clearly a much better approximation and seems to be behaving as expected: the approximation matches the exact solution at the nodes and the now curved piecewise function seems to match the curve much closer between the nodes.

9 Implementing higher order models

9.1 Higher order models

9.1.1 Basis functions

In order to increase the order even higher, the same process is utilised as before. All that is needed is a standard prescription.

As before, it would make sense to, for the pth order polynomial, produce a pth order base function which can be rescaled to each element as before.

The basis chosen is the simple construction:

$$\hat{\phi}_{\mathsf{p}}(\eta) = \begin{cases} \prod_{k=0}^{p-1}(-1)^{p-1}\left(\eta - \frac{k}{p-1}\right) & \text{if } 0 \leq \eta \leq 1, \\ 0 & \text{otherwise} \end{cases} \tag{9.1}$$

i.e. on the interval $[0, 1]$, the pth order polynomial base function is given p evenly spaced roots. The $(-1)^{p-1}$ term is just put in to make the functions orient the same way

Note that this is consistent with the quadratic base function already established in (8.1)

It is also important to note that this is just one possible choice of basis. For example, the basis demonstrated in Schwab [16] is based on legendre polynomials, utilising their orthonormality to make the function more efficient. This basis was chosen for its simple construction and manipulation in the code but it's possible than an alternate choice would've been preferable.

Then, as before, the pth order contribution on the element $[x_k, x_{k+1}]$ is given by:

$$\phi_{\mathsf{k+(p\text{-}1)N}}(x) = \begin{cases} \hat{\phi}_{\mathsf{p}}\left(\frac{x-x_k}{h_k}\right) = \prod_{k=0}^{p-1}(-1)^{p-1}\left(\frac{x}{h_k} - \frac{x_k}{h_k} - \frac{k}{p-1}\right) & \text{if } x_k \leq x \leq x_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{9.2}$$

9.1.2 Corresponding model

It is simple to then construct the model at pth order as:

$$
\begin{aligned}
u_h(x) &= \sum_{k=1}^{pN-1} u_k\phi_k(x) + A\phi_0(x) + B\phi_{pN}(x) \\
&= \sum_{n=1}^{p} u_0^n(x) + u^1_{D(A,B)}(x)
\end{aligned} \tag{9.3}
$$

9.1.3 Confirming space consistency

As for previous models, it needs to be shown that the new model is consistent with the space $V^p \subseteq H^1_{D(A,B)}([a,b])$ which is defined here as:

$V^p = span\{\phi_1 \cdots \phi_{pN-1}\} + A\phi_0 + B\phi_{pN}$

Since it was already proved in section 7.1.2 that $u^1_{h,0}(x) \in H^1_0([a,b])$ and $u^1_{h,D(A,B)}(x) \in H^1_0([a,b])$ and in 8.1.3 that $u^2_{h,0}(x) \in H^1_0([a,b])$, all that would be required to prove $u_h(x) = \sum_{k=0}^{pN} u_k\phi_k(x) \in V^p \subseteq H^1_{D(A,B)}([a,b])$ is to show, if $u_h^{(p-1)}(x) = \sum_{k=0}^{N-1} u_k\phi_{k+(p-2)N}(x) \in H^1_0([a,b])$ then $u^p_h(x) = \sum_{k=0}^{N-1} u_k\phi_{k+(p-1)N}(x) \in H^1_0([a,b])$.

If this is proved, then, by an inductive argument, $u^p_h(x) \in H^1_0([a,b]) \forall p \in \mathbb{N}$

This can be done in the following steps:

1) Show $u^p_h \in L^2([a,b])$

Recalling the definition in (2.4), this result is equivalent to proving:

Show $||u^p_h||_{L^2([a,b])} = \left(\int_a^b |u^p_h(x)|^2 dx\right)^{\frac{1}{2}} < \infty$

Due to the fact that these bases don't overlap, this can simplified to:

$\left(\int_a^b |u^p_h(x)|^2 dx\right)^{\frac{1}{2}} = \left(\sum_{k=0}^{N-1} |u_{k+pN}|^2 \int_{x_k}^{x_{k+1}} \phi_{k+pN}(x)dx\right)^{\frac{1}{2}}$

Since all $\phi_{k+(p-1)N}$ are continuous on the interval, these integrals will all be finite so:

$||u^p_h||_{L^2([a,b])} = \left(\sum_{k=0}^{N-1} |u_{k+(p-1)N}|^2 \int_{x_k}^{x_{k+1}} \phi_{k+(p-1)N}(x)dx\right)^{\frac{1}{2}} \leq \infty$

2) Calculate $D_1(\phi_{k+(p-1)N}(x))$

Recalling the definition in (2.5), this means:

Find $D_1(\phi_{k+(p-1)N}(x))$ s.t. $\int_a^b \phi_{k+(p-1)N}(x)v'(x)dx = -\int_a^b D_1(\phi_{k+(p-1)N}(x))v(x)dx$

Starting from the left hand side, the following manipulations can be made:

$$
\begin{aligned}
\int_a^b \phi_{k+N}(x)v'(x)dx &= \int_{x_k}^{x_{k+1}} \phi_{k+(p-1)N}(x)v'(x)dx \\
&= \Big[\phi_{k+(p-1)N}(x)v(x)\Big]_{x_{k-1}}^{x_k} - \int_{x_{k-1}}^{x_k} \phi'_{k+(p-1)N}(x)v(x)dx \\
&= -\int_{x_{k-1}}^{x_k} \phi'_{k+(p-1)N}(x)v(x)dx
\end{aligned}
$$

By inspection, it can clearly be seen that the derivative is therefore given by the point-wise derivative of the function:

$$D_1(\phi_{k+(p-1)N}(x)) = \begin{cases} \phi'_{k+(p-1)N}(x) & x_k < x < x_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{9.4}$$

3) Show $D_1(u_h(x)) \in L^2([a,b])$

By (9.4), it can clearly be seen that:

$D_1(u_h^p(x)) = \sum_{k=0}^{N-1} u_k D_1(\phi_{k+(p-1)N}(x))$

is a piecewise (p-1) order function on the mesh. This is clearly a member of $L^2([a,b])$ as, similarly to 7.1.2, it can be shown that the L^2 norm of the function is just a finite sum of finite integrals given the function is continuous on the intervals.

4) Show $u_h^p(a) = u_h^p(b) = 0$

This is trivial as this is a condition of the basis setup here.

Summary

By the arguments above, it has therefore been confirmed that:

If $u_h^{(p-1)}(x) \in H_0^1([a,b])$ then $u_h^p(x) \in H_0^1([a,b])$. Since $u_h^1(x), u_h^2(x) \in H_0^1([a,b])$ and $u_h^1(x) \in H^1_{D(A,B)}([a,b])$, by induction:

$u_h(x) = \sum_{k=0}^{pN} u_k \phi_k(x) \in V_h^p \subseteq H^1_{D(A,B)}([a,b])$

Therefore the new basis is consistent with the previous basis for any arbitrary order of polynomial so this is a valid construction.

9.2 Higher order linear system

9.2.1 Higher order system prescription

As before, it is easy to see that the finite element system is now a $(pN-1) \times (pN-1)$ system prescribed by:

Find $u_j \in \mathbb{R}$ s.t.

$$\sum_{j=1}^{pN-1} u_j a(\phi_j(x), \phi_i(x)) = l(\phi_i(x)) - A \cdot a(\phi_0(x), \phi_i(x)) - B \cdot a(\phi_{pN}(x), \phi_i(x)) \qquad \forall i \in [1, \ldots, pN-1] \tag{9.5}$$

Which rearranges to the same matrix system as in 8.6

9.2.2 Setting up the higher order linear system

A general element may now look something like:

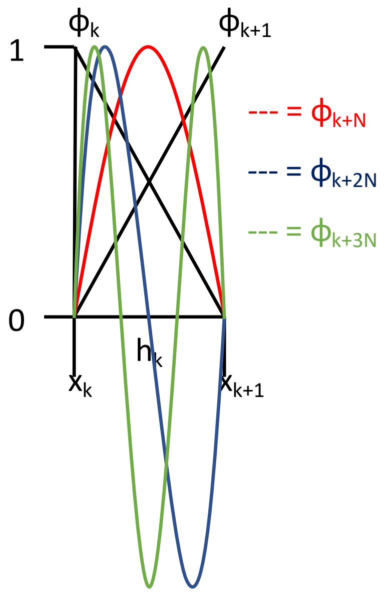

Figure 9: 4th order basis element

Iterating element-wise, the non zero contributions to the matrix are found at the kth element at matrix elements $A_{i,j} \forall i, j \in [k, k+1, k+N, \ldots, k+PN]$

Utilising the same method as demonstrated in 1 adding in the new interactions in the integrals gives the following algorithm:

```
1: procedure Full linear system setup pth order
       $\underline{u(a) = A}$
2:     $A_{0,0} = 1$
       $L_0 = A$
3:     for all $k \in [0 : N-1]$ do
```

```
4:     if $k \neq 0$ and $k \neq N-1$ then
           $\underline{A_{k+1,k}}$
           $A_{k+1,k} = A_{k+1,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
           $\underline{A_{k,k+1}}$
           $A_{k,k+1} = A_{k,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
5:     else if $k \neq 0$ then
           $\underline{A_{k,k}}$
           $A_{k,k} = A_{k,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
6:         for all $i \in [1:p-1]$ do
           $\underline{A_{k+iN,k}}$
           $A_{k+iN,k} = A_{k+iN,k} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}'_{\text{down}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\text{down}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta$
           $\underline{A_{k,k+iN}}$
           $A_{k,k+iN} = A_{k,k+iN} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{down}}(\eta)\hat{\phi}'_{i+1}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{i+1}(\eta)\hat{\phi}_{\text{down}}(\eta)d\eta$
7:         end for
8:     else if $k \neq N-1$ then
           $\underline{A_{k+1,k+1}}$
           $A_{k+1,k+1} = A_{k+1,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{\text{up}}(\eta)d\eta$
9:         for all $i \in [1:p-1]$ do
           $\underline{A_{k+iN,k+1}}$
           $A_{k+iN,k+1} = A_{k+iN,k+1} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}'_{\text{up}}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{\text{up}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta \quad +nh_k\int_0^1 \hat{\phi}_{\text{up}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta$
```

```
            $\underline{A_{k+1,k+iN}}$
            $A_{k+1,k+iN} = A_{k+1,k+iN} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}'_{i+1}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta$
                $+nh_k\int_0^1 \hat{\phi}_{i+1}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta$
10:         end for
11:     else
12:         for all $i \in [1 : p-1]$ do
            $\underline{A_{k+iN,k+iN}}$
            $A_{k+iN,k+iN} = A_{k+iN,k+iN} + \frac{1}{h_k}\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}'_{i+1}(\eta)d\eta \quad +m\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}_{i+1}(\eta)d\eta$
                $+nh_k\int_0^1 \hat{\phi}_{i+1}(\eta)\hat{\phi}_{i+1}(\eta)d\eta$
13:         end for
14:     end if
15:     if $k \neq 0$ then
            $\underline{L_k}$
            $L_k = L_k + \int_{x_{k-1}}^{x_{k+1}} f(x)\phi_k(x)dx$
16:     end if
17: end for
    $\underline{u(b) = B}$
18: $A_{N,N} = 1$
    $L_N = B$
    Boundary corrections
19: $L_1 = L_1 - A\left(\frac{1}{h_0}\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}'_{\mathsf{down}}(\eta)d\eta\right.$
        $\left. +m\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta + nh_0\int_0^1 \hat{\phi}_{\mathsf{down}}(\eta)\hat{\phi}_{\mathsf{up}}(\eta)d\eta\right)$
20: for all $i \in [1 : p-1]$ do
     $L_{iN} = L_{iN} - A\left(\frac{1}{h_0}\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}'_{\mathsf{down}}(\eta)d\eta\right.$
        $\left. +m\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta + nh_0\int_0^1 \hat{\phi}_{\mathsf{down}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta\right)$
21: end for
```

```
22:     $L_{N-1} = L_{N-1} - B\left(\frac{1}{h_{N-1}}\int_0^1 \hat{\phi}'_{\mathsf{down}}(\eta)\hat{\phi}'_{\mathsf{up}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta + nh_{N-1}\int_0^1 \hat{\phi}_{\mathsf{up}}(\eta)\hat{\phi}_{\mathsf{down}}(\eta)d\eta\right)$
23:     for all $i \in [1:p-1]$ do
            $L_{(i+1)N-1} = L_{(i+1)N-1} - B\left(\frac{1}{h_{N-1}}\int_0^1 \hat{\phi}'_{i+1}(\eta)\hat{\phi}'_{\mathsf{up}}(\eta)d\eta + m\int_0^1 \hat{\phi}'_{\mathsf{up}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta + nh_{N-1}\int_0^1 \hat{\phi}_{\mathsf{up}}(\eta)\hat{\phi}_{i+1}(\eta)d\eta\right)$
24:     end for

25:     return $A, L$
26: end procedure
```

Applied properly, this should give a linear system with sparsity pattern looking like:

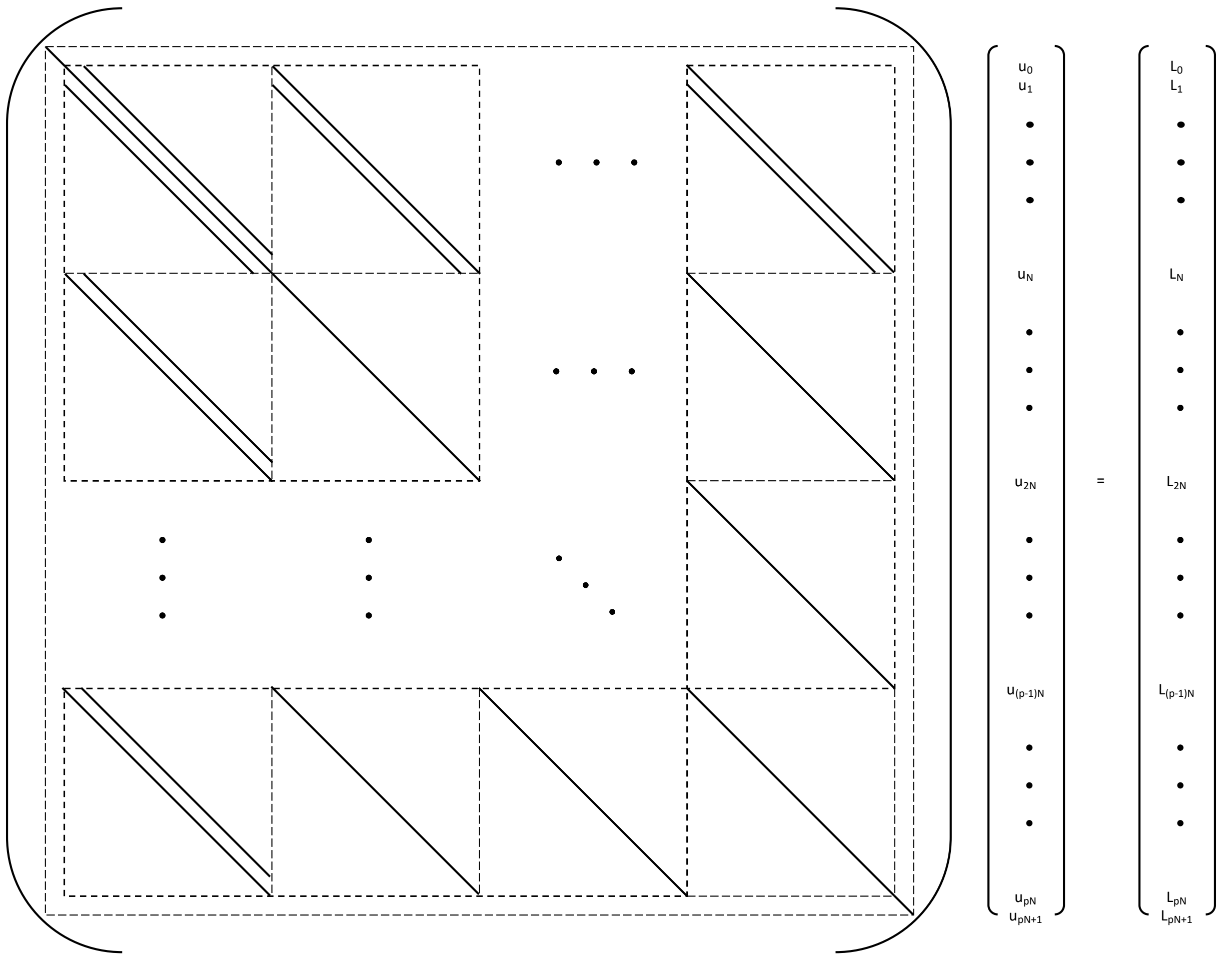

Figure 10: General sparsity pattern at pth order

9.3 Higher order approximation numerical example

Working with the same model as before as given in 7.15 at cubic order yields the following output:

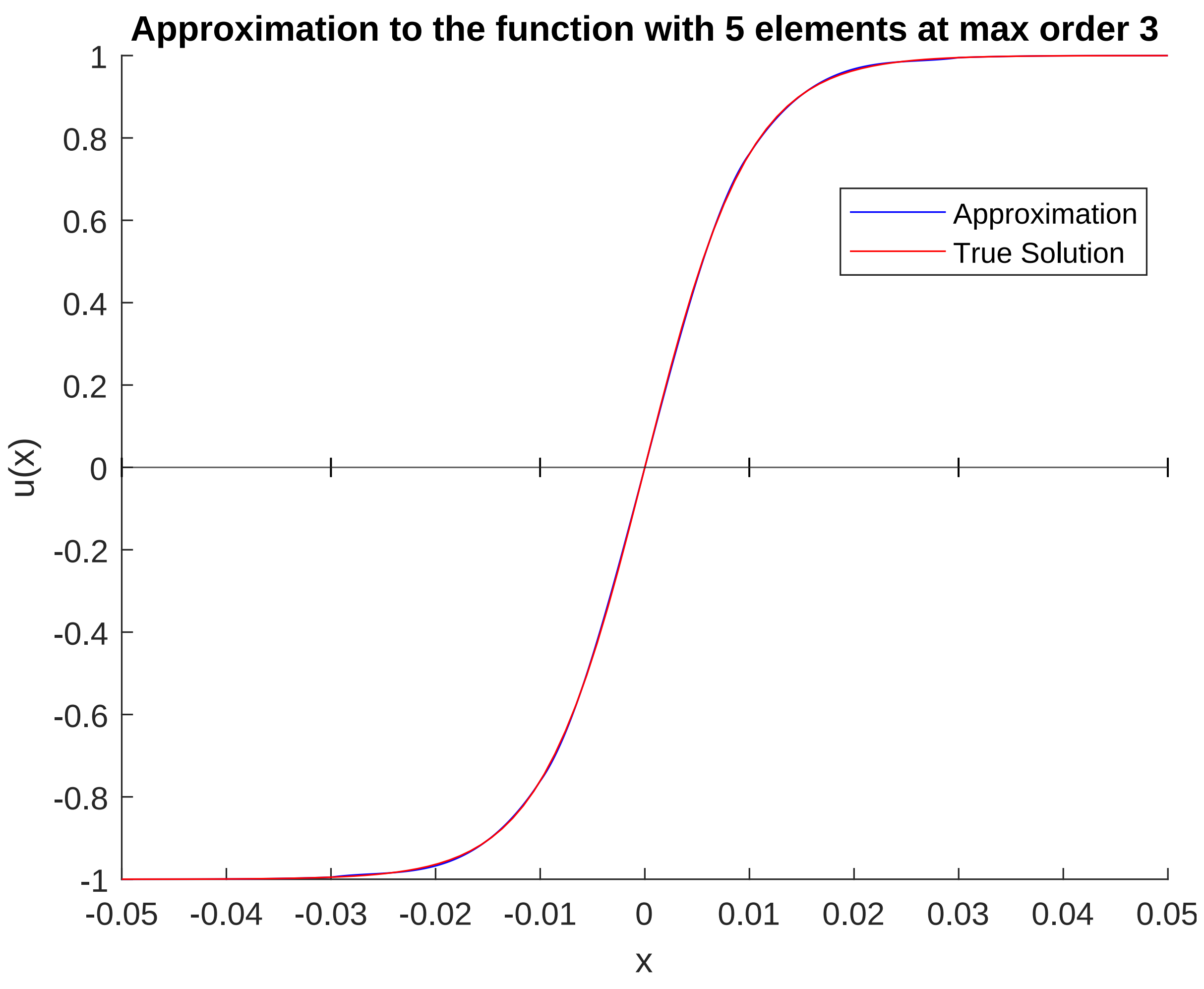

Figure 11: Cubic approximation in action

Which already looks barely distinguishable from the true solution. Higher order approximations give more accurate outputs but are not included here as they are visually indistinguishable from each other.

Clearly this method is very powerful but this isn't utilising the full power of the finite element method as the scheme is applying a uniform order on a uniform mesh.

10 Order of convergence

It has been demonstrated in figures 4, 8 and 11 that the models look very similar to the approximations, particularly as the order is increased. This isn't very mathematically rigorous though so additional confirmation is sought that the method is being applied correctly.

10.1 General result

Expanding upon the result in (6.9), it can be shown that for sufficiently smooth solution functions (i.e. functions in the space $H^{p+1}([a,b])$ where the approximation is of order p), the following orders of convergence can be established:

For the pth order approximation u_h to a solution function u, $\exists K \in \mathbb{R}$ s.t:

$$||u - u_h||_{H^1([a,b])} \leq Kh^p|u|_{H^{p+1}([a,b])} \tag{10.1}$$

$$||u - u_h||_{L^2([a,b])} \leq Kh^{p+1}|u|_{H^{p+1}([a,b])} \tag{10.2}$$

A full proof of this result can be found in Ciarlet (2002,Thm 3.2.2/Thm 3.2.5.) [9] as well as Ern and Guermond (2004,Proposition 1.12) [13].

The proof is too involved to write out in full here as it will just be used as a sanity check that the algorithm is behaving as expected.

10.2 Verifying approximation convergence

Since the result specified a sufficiently smooth function, the test problem for order of convergence will be the simple poisson equation:

$$u''(x) = \cos(\pi x) \qquad x \in [0,1] \tag{10.3}$$

If this function is approximated using the approximation of a fixed constant order at several meshes of varying interval widths, these interval widths can be plotted against their resultant errors in the following log-log plot:

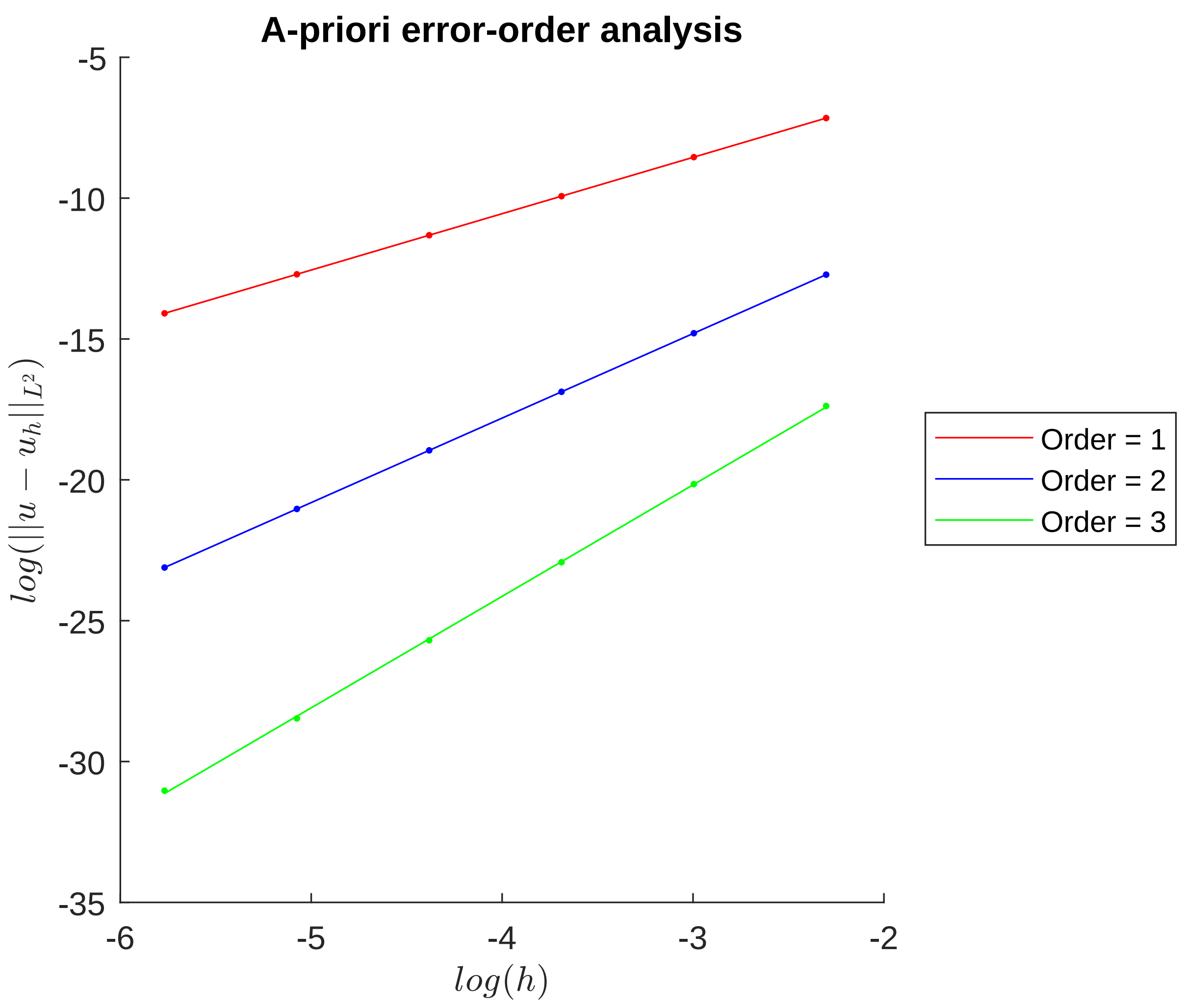

Figure 12: Approximation convergence analysis

where clearly, the results that higher order results in lower error can be seen to be replicated here.

The orders can then be found by finding the gradient of these plots:

Approximation order	**Order of L_2 errors w.r.t h (2d.p)**
1	2.00
2	3.00
3	3.96

Table 1: Error order analysis

These results line up quite well with the theory, implying that the method has indeed been applied correctly. There is a slight drift from the expected value as the order of approximation increases but this is likely due to factors like the Runge phenomenon coming into play:

Definition 10.1. Runge phenomenon

This is a well known phenomenon [23] whereby polynomial approximations to a function with a discontinuity or sharp corner develop oscillations or overshoots, particularly for higher order polynomials where the oscillations become more pronounced and difficult to control.

This can manifest for the piecewise polynomial approximations used here at the element boundaries especially as these are non smooth. Although a greater number of elements will mitigate this issue as the error contributions from each element will be lessened.

11 Non-uniform hp systems

So far, all approximation models implemented have had a consistent order across the whole domain and a constant interval width but there is no reason that this needs to be the case.

11.0.1 Inconsistent interval width

Although it hasn't been implemented to show it, the algorithms written already account for discretisation of domains into elements where the interval width is not necessarily a constant.

The kth element is denoted as having interval width h_k. All that needs to be done is to keep a record of where the nodal points are as it is trivial to extract the corresponding interval widths from this

11.0.2 Inconsistent order

Similarly, all models implemented so far have had the same order of approximation across the whole domain but again, there's no reason that this has to be the case.

Similar to interval width, this is dealt with simply by establishing a vector of desired orders at each element.

Now it can be clearly seen why element-wise iteration was desirable for the algorithm. Iterating in this way means that, at each element, the order here can be extracted and contributions up to this order can be added to the matrix.

There is a slight complication in that a track has to be taken of how many elements are at each order and where they are located in the domain to know where to put new elements and understanding the meaning of coefficients but this is relatively easily done.

11.0.3 Consistency with previous theory

This non-uniform system trivially still belongs to the space $H^1_{D(A,B)}([a,b])$ as it is equivalent to the function of consistent order at the maximum order in the vector just with some of the coefficients set to be zeroes.

12 Error estimation

The ability to form systems as discussed in section 11 means that an adaptive method can be implemented. In order to do this however, the algorithm needs to know where the error in the approximation is worst so refinements can be carried out in the correct regions.

This error can be calculated in two main ways:

12.1 A-priori error

The most obvious and easy way to calculate error is a-priori error. This is calculated as some norm of the difference between the known solution and the approximation calculated. In this case, the one that will be used is the L2 norm: $||u - u_h||_{L^2([a,b])}$

The obvious problem with this is that the computation requires prior knowledge of what the actual solution is. This makes this error calculation useful for assessing how well the solution or its error are being approximated in cases where the solution is known but useless as an error estimate for when unknown systems are being solved.

It is now necessary to come up with some way of estimating the error without using the known solution.

12.2 Residual based error estimates

12.2.1 Residual definition

By inspection, it may make sense to base the error estimate on the residual of the problem with the following definition:

For the problems as given previously in equation (4.4) where there is a true solution and an approximation defined:

Definition 12.1. True solution

$\exists u \in H_0^1$ s.t.

$a(u, v) = l(v) \qquad \forall v \in H_0^1$

Definition 12.2. Approximation

$\exists u_h \in V_h$ s.t.

$a(u_h, v_h) = l(v_h) \qquad \forall v_h \in V_h$

The residual is then defined based on this as:

Definition 12.3. Residual definition

Let $e_h = u - u_h$, then:

$$\begin{aligned} a(e_h, v) &= a(u, v) - a(u_h, v) \\ &= l(v) - a(u_h, v) \\ &= R(v) \end{aligned}$$

i.e.

$R(v) = l(v) - a(u_h, v)$

(It is noted that this definition implies $R(v_h) = 0 \forall v_h \in V_h$)

This can be intuitively seen as a measure of how well the approximation solves the original problem posed with the residual approaching zero as the approximation approaches the true solution. Notably, the residual requires no knowledge of the true solution in its calculation. Therefore, if it can be used to approximate the error, this would be known as an 'a-posteriori' error estimate, much preferable to the a-priori errors discussed in section 12.1

A more mathematical demonstration that this is indeed a measure of the error is sought:

12.2.2 Demonstrating the validity of a residual based error bound

From the coercivity result proved in 4.5.3, it is known that:

$a(v, v) \geq c_0 ||v||^2_{H^1} \qquad \forall v \in H^1_0([a, b])$

Applying this result to $e_h \in H^1_0([a, b])$ gives:

$$a(e_h, e_h) = R(e_h) \geq c_0 ||e_h||^2_{H^1} \tag{12.1}$$

so the residual can be used to bound the error norm.

12.3 Derivation of residual based error estimate

Using the concept of the residual as a starting point, an error estimate can be found. Here the derivation is given for the Poisson's equation:

12.3.1 Derivation for Poisson's equation

Definition 12.4. Strong form of Poisson's equation

$$-u''(x) = f(x) \qquad x \in [a,b]$$

where: $u(a) = u(b) = 0$

Which can be seen to just be a simplification of the more general problem being tackled in (3.1). The results shown here generalise to this equation, the extra terms incurred are just omitted for expedience.

The above system can be rewritten as the bilinear expression of the weak form where the following are satisfied:

Definition 12.5. True solution

The true solution $u_0(x) \in H_0^1([a,b])$ satisfies:

$$a(u_0, v) = l(v) \qquad \forall v(x) \in H_0^1([a,b])$$

where $a(u,v) = \int_a^b v'(x)u'(x)dx$

for $l(v) = \int_a^b f(x)v(x)dx$

An approximation $u_h(x) \in V_h \subseteq H_0^1([a,b])$ is defined on some mesh Ω with N elements with orders $\underline{p}$

Definition 12.6. Approximation

The approximation $u_h(x) \in V_h$ satisfies:

$$a(u_h, v_h) = l(v_h) \qquad \forall v_h(x) \in V_h$$

For which the residual is defined:

Definition 12.7. Residual

$$\begin{aligned} R(v) &= l(v) - a(u_h, v) \\ &= \int_a^b f(x)v(x)dx - \int_a^b u_h'(x)v(x)dx \end{aligned}$$

This definition can then be manipulated in the following way:

$$\begin{aligned} R(v) &= \int_a^b f(x)v(x)dx - \int_a^b u_h'(x)v(x)dx \\ &= \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} f(x)v(x)dx - \int_{x_{j-1}}^{x_j} u_h'(x)v(x)dx \right) \\ &= \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} f(x)v(x)dx - \left([u_h'(x)v(x)]_{x_{j-1}}^{x_j} - \int_{x_{j-1}}^{x_j} u_h''(x)v(x)dx \right) \right) \end{aligned}$$

(by integration by parts)

The term $\sum_{j=1}^{N}[u_h'(x)v(x)]_{x_{j-1}}^{x_j}$ can then be explicitly evaluated. v is a continuous function but the derivative of u_h is not necessarily.

Letting $u_{h,R}'$ be the right handed limit of the function and $u_{h,L}'$ denote the left handed limit of the function, the evaluation is given:

$$\begin{aligned} \textstyle\sum_{j=1}^{N}[u_h'(x)v(x)]_{x_{j-1}}^{x_j} = \quad & u_{h,L}'(x_1)v(x_1) && -0 \\ & +u_{h,L}'(x_2)v(x_2) && -u_{h,R}'(x_1)v(x_1) \\ & +u_{h,L}'(x_3)v(x_3) && -u_{h,R}'(x_2)v(x_2) \\ & \vdots && \vdots \\ & +u_{h,L}'(x_{N-1})v(x_{N-1}) && -u_{h,R}'(x_{N-2})v(x_{N-2}) \\ & +0 && -u_{h,L}'(x_{N-1})v(x_{N-1}) \end{aligned}$$

Where the boundary conditions for the function v have been used.

Letting $[u_h]'(x) = u_{h,R}'(x) - u_{h,L}'(x)$ denote the jump in the derivative at a point 'x', the sum can be rewritten:

$\sum_{j=1}^{N}[u_h'(x)v(x)]_{x_{j-1}}^{x_j} = \sum_{j=1}^{N-1}\left([u_h]'(x_j)v(x_j)\right)$

which makes the full evaluation:

$R(v) = \sum_{j=1}^{N}\left(\int_{x_{j-1}}^{x_j} f(x)v(x) + u_h''(x)v(x)dx\right) - \sum_{j=1}^{N-1}\left([u_h]'(x_j)v(x_j)\right)$

The following part of this proof relies on a projection operator as defined in Theorem 3.14 of [16]. This projects v onto the same vector space as u_h (i.e. the space of a continuous piecewise polynomial with variable order across a given mesh), satisfying certain additional properties. It is denoted $\pi_{\underline{p}}v \in V_h$

From the property $\pi_{\underline{p}}v \in V_h$, it can be concluded that $R(\pi_{\underline{p}}v) = 0$

Hence:

$$R(v) = R(v - \pi_{\underline{p}} v) = \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} (f(x) + u_h''(x))(v(x) - \pi_{\underline{p}} v(x)) dx \right)$$
$$- \sum_{j=1}^{N-1} \left([u_h]'(x_j)(v(x_j) - \pi_{\underline{p}} v(x_j)) \right)$$

The projection $\pi_{\underline{p}} v(x)$ is defined in Theorem 3.17 of [16] to match its projected function v at the boundary points of each interval. This eliminates the final term.

A slightly different projection is applied to the function f which projects it to the space of discontinuous piecewise polynomials defined on the same mesh and order vector as V_h. It is constructed as the element-wise Legendre series of f to the appropriate order: $\pi_{\underline{s}} f(x) \in V_h'$ s.t. $V_h \subset V_h'$

This projection therefore has the property:

$\langle \pi_{\underline{s}} f, v \rangle = \langle f, v \rangle \qquad \forall v \in V_h'$

where $\langle v, w \rangle = \int_a^b v(x) w(x) dx$ is the L^2 projection

Hence:

$\langle \pi_{\underline{s}} f - f, v \rangle = 0 \qquad \forall v \in V_h'$

Since $V_h \subset V_h'$, this also holds $\forall v \in V_h$, the following rearrangement can then be made:

$\langle \pi_{\underline{s}} f - f, \pi_{\underline{p}} v \rangle = 0 = \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} (\pi_{\underline{s}} f(x) - f(x)) \pi_{\underline{p}} v(x) dx \right)$

which can therefore be substituted back into the equation for the residual:

$R(v) = \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} (f(x) + u_h''(x))(v(x) - \pi_{\underline{p}} v(x)) dx - \int_{x_{j-1}}^{x_j} (\pi_{\underline{s}} f(x) - f(x))(\pi_{\underline{p}} v(x)) dx \right)$

Since

$$f(x) = f(x) + \pi_{\underline{s}} f(x) - \pi_{\underline{s}} f(x)$$
$$= \pi_{\underline{s}} f(x) + (f - \pi_{\underline{s}} f(x))$$

this can be substituted into the residual expression to give:

$$R(v) = \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} (f(x) + u_h''(x))(v(x) - \pi_{\underline{p}} v(x)) dx \right.$$
$$\left. + \int_{x_{j-1}}^{x_j} (\pi_{\underline{s}} f(x) - f(x))(v(x)) dx - \int_{x_{j-1}}^{x_j} (\pi_{\underline{s}} f(x) - f(x))(\pi_{\underline{p}} v(x)) dx \right)$$
$$= \sum_{j=1}^{N} \left(\int_{x_{j-1}}^{x_j} (\pi_{\underline{s}} f(x) + u_h''(x))(v(x) - \pi_{\underline{p}} v(x)) dx \right.$$
$$\left. + \int_{x_{j-1}}^{x_j} (\pi_{\underline{s}} f(x) - f(x))(v(x) - \pi_{\underline{p}} v(x)) dx \right)$$

A weight function $w_j(x)$ is then introduced, defined as:

$w_j(x) = (x_j - x)(x - x_{j-1})$

This can be incorporated into the expression to apply the Cauchy-Schwarz inequality 2:

$$R(v) = \sum_{j=1}^{N}\left(\int_{x_{j-1}}^{x_j}((\pi_{\underline{s}}f(x)+u_h''(x))\sqrt{w_j(x)})\left(\frac{v(x)-\pi_{\underline{p}}v(x)}{\sqrt{w_j(x)}}\right)dx\right.$$

$$\left.+\int_{x_{j-1}}^{x_j}((\pi_{\underline{s}}f(x)-f(x))\sqrt{w_j(x)})\left(\frac{v(x)-\pi_{\underline{p}}v(x)}{\sqrt{w_j(x)}}\right)dx\right)$$

$$\leq \sum_{j=1}^{N}\left(\left(\int_{x_{j-1}}^{x_j}((\pi_{\underline{s}}f(x)+u_h''(x))^2 w_j(x)dx\right)^{\frac{1}{2}}\left(\int_{x_{j-1}}^{x_j}\frac{(v(x)-\pi_{\underline{p}}v(x))^2}{w_j(x)}dx\right)^{\frac{1}{2}}\right.$$

$$\left.+\left(\int_{x_{j-1}}^{x_j}((\pi_{\underline{s}}f(x)-f(x))^2 w_j(x)dx\right)^{\frac{1}{2}}\left(\int_{x_{j-1}}^{x_j}\frac{(v(x)-\pi_{\underline{p}}v(x))^2}{w_j(x)}dx\right)^{\frac{1}{2}}\right)$$

From Theorem 3.3.17 in [16], the projection has the following property:

$$\int_{x_{j-1}}^{x_j}\frac{(v(x)-\pi_{\underline{p}}v(x))^2}{w_j(x)}dx \leq \frac{1}{p_j(p_j+1)}\int_{x_{j-1}}^{x_j}(v'(x))^2dx \tag{12.2}$$

Substituting this back into the evaluation of the residual yields:

$$R(v) \leq \sum_{j=1}^{N}\left(\left(\int_{x_{j-1}}^{x_j}((\pi_{\underline{s}}f(x)+u_h''(x))^2 w_j(x)dx\right.\right.$$

$$\left.\left.+\int_{x_{j-1}}^{x_j}((\pi_{\underline{s}}f(x)-f(x))^2 w_j(x)dx\right)\left(\frac{1}{p_j(p_j+1)}\int_{x_{j-1}}^{x_j}(v'(x))^2dx\right)\right)^{\frac{1}{2}}$$

$$=\left(\sum_{j=1}^{N}\left(\frac{1}{p_j(p_j+1)}\left(||r_j w_j^{\frac{1}{2}}||^2_{L^2(\Omega_j)}+||(\pi_{\underline{s}}f-f)w_j^{\frac{1}{2}}||^2_{L^2(\Omega_j)}\right)\right)^{\frac{1}{2}}\right)||v'||_{L^2(\Omega)}$$

where: $r_j(x)=\pi_{\underline{s}}f(x)+u_h''(x)$ is the element residual on each element Ω_j

This defines the residual based error estimate:

Definition 12.8. Residual based error estimate

$(EST)^2 = \sum_{j=1}^{N}\eta_j^2 + \frac{1}{p_j(p_j+1)}||(\pi_{\underline{s}}f-f)w_j^{\frac{1}{2}}||^2_{L^2(\Omega_j)}$

where $\eta_j^2 = \frac{1}{p_j(p_j+1)}||r_j w_j^{\frac{1}{2}}||^2_{L^2(\Omega_j)}$

and $r_j(x)=\pi_{\underline{s}}f(x)+u_h''(x)$

for which the inequality:

$$R(v) \leq (EST)||v'||_{L^2} \tag{12.3}$$

holds

12.3.2 Proof of validity of error estimate

The result in (12.1) can now be extended with (12.3) to give:

$$\begin{aligned} c_0||e_h||_{H^1}^2 \le R(e_h) &\le (EST)||e_h'||_{L^2} \\ &\le (EST)||e_h||_{H^1} \end{aligned} \tag{12.4}$$

Rearranging this gives the result:

$$||e_h||_{H^1} \le \frac{1}{c_0}(EST) \tag{12.5}$$

proving that true error is indeed bounded by this residual-based approximation to it.

12.4 Explicit residual based error estimate

For the slightly more complex problem being tackled in this project, the definition of the error estimate can be seen to generalise to:

Definition 12.9. Residual based error estimate general
$(EST)^2 = \sum_{j=1}^{N} \eta_j^2 + \frac{1}{p_j(p_j+1)}||(\pi_{\underline{s}} f - f)w_j^{\frac{1}{2}}||_{L^2(\Omega_j)}^2$
where $\eta_j^2 = \frac{1}{p_j(p_j+1)}||r_j w_j^{\frac{1}{2}}||_{L^2(\Omega_j)}^2$
and $r_j(x) = \pi_{\underline{s}} f(x) + u_h''(x) - m \cdot u_h'(x) - n \cdot u_h(x)$

To which the following simplifications are made:

1. The weight function $w_j(x) = (x_j - x)(x - x_{j-1})$ is replaced with the simple constant h_{j-1}^2. It can clearly be seen that $(x_j - x)(x - x_{j-1}) \le h_{j-1}^2$ so the error bounds will hold, the approximation may just be slightly less accurate.

2. The two functions $\pi_{\underline{s}} f(x)$ and $f(x)$ are regarded as the same. This is not strictly accurate, particularly for initial approximations but as the mesh shrinks and the orders get higher, the projection of f onto this system will approach the true value of f. In this way, the approximations to the error may start as being slightly inaccurate but, as the iterations continue, this factor should become less and less important. Since high precision is only sought for the later iterations, this seems a valid choice.

The simplified prescription is then given:

Definition 12.10. Practical residual based error estimate

$(EST)^2 = \sum_{j=1}^{N} \eta_j^2$

where $\eta_j^2 = \frac{h_j^2}{p_j(p_j+1)} ||r_j||^2_{L^2(\Omega_j)}$

and $r_j(x) = f(x) + u_h''(x) - m \cdot u_h'(x) - n \cdot u_h(x)$

12.5 Demonstration of the error estimate

This can be seen in action applied to a consistent cubic order approximation over 100 evenly spaced elements:

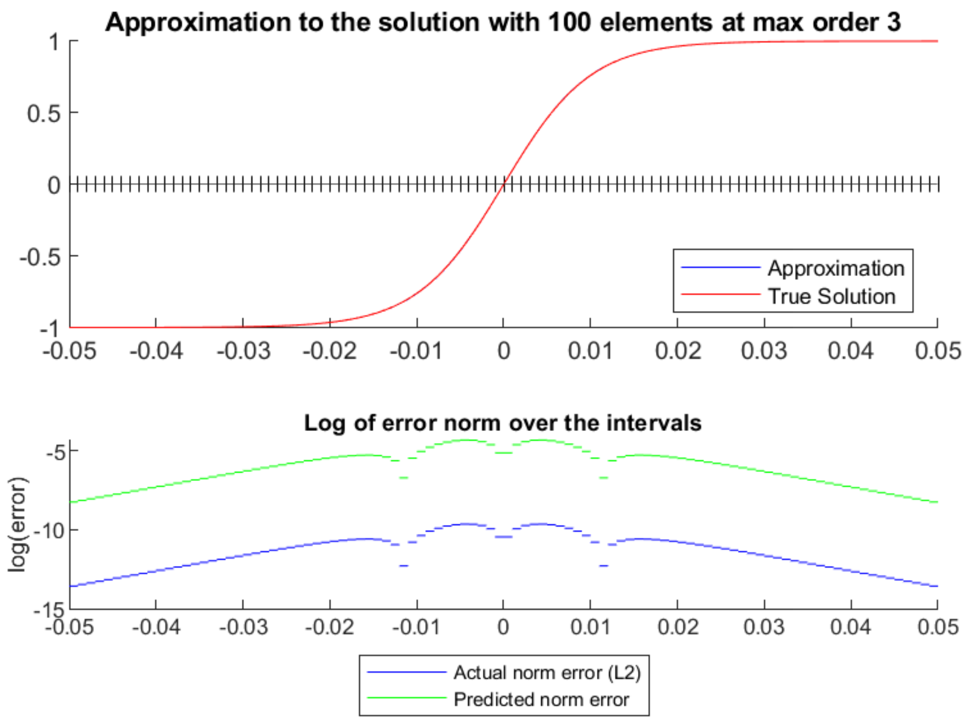

Figure 13: A-priori/a-posteriori comparison

As the plot clearly demonstrates, although the predicted error is much higher than the actual error, the errors follow the exact same pattern over the domain.

This means that the approximation used will be very effective at identifying where the areas which need the most refinement can be located.

Only one graph is shown here but this result has been consistently demonstrated for a range of functions.

13 Adaptivity algorithm

13.1 Determining tolerance at which to flag elements

Now that there is an estimate for the error on each interval, the algorithm requires that some condition is given for which of these errors are large enough to refine and which should be left alone for this iteration.

There are two main ways in which this could be done:

1) Equi-distributed tolerance

A standard approach would be to just state some global tolerance that the model should reach.

This global tolerance could then be distributed across the intervals as a function of their interval width with respect to the whole domain:

This would give a local tolerance that, if all intervals were at this tolerance, the whole domain could be guaranteed to be below the global tolerance.

e.g. on the interval $[x_k, x_{k+1}]$:

$$\mathsf{TOL}_{\mathsf{LOCAL}} = \frac{h_k}{b-a} \cdot \mathsf{TOL}_{\mathsf{GLOBAL}} \tag{13.1}$$

This would be desirable if the error estimation was very accurate but, as was made clear in figure 13, the actual values of error and error estimates are quite far apart so the global error could definitely be guaranteed but the actual error would be much lower than reported meaning more computation than was required would've been carried out.

2) Percentage reduction

A method that is more desirable in this case is percentage reduction.

This works by reducing the top pth percentile of interval errors on the domain at each iteration.

i.e. for the current set of errors: $\underline{E}$:

$$\mathsf{TOL}_{\mathsf{LOCAL}} = \text{percentile}(\underline{E}, p) \tag{13.2}$$

In this way, each iteration reduces only the worst errors on the domain so over time the whole error profile will be reduced in an efficient manner.

This is the method selected for the algorithm therefore.

13.2 Stopping criteria

Section 13.1 provides a means to run the algorithm but it can't just run indefinitely. There needs to be a point at which it is stopped. Again, there are a few ways to do this:

1) Set number of iterations

The simplest approach would be to just run the algorithm n times.

i.e. Let the maximum number of iterations be 'N', then the stopping criteria is given:

```
1: procedure STOPPING CRITERIA ITERATION LIMIT
       iterations = 0
2:     while iterations<N do
           Iterate adaptive algorithm
           iterations = iterations+1
3:     end while
4: end procedure
```

It would be known at this point that the algorithm had been refined n times and therefore that the error had been reduced somewhat from the initial mesh in-putted.

From there, the error could be assessed and the algorithm can be run another n times until the error is at an acceptable level

2) Global error tolerance

The alternative approach is to set a desired global tolerance and iterate until the reported global error comes back as under this value.

i.e. Let the current estimated global error be written $\text{error}_{\text{global}}$ and the corresponding tolerance $\text{TOL}_{\text{GLOBAL}}$, then the stopping criteria is given:

```
1: procedure STOPPING CRITERIA TOLERANCE LIMIT
2:     while $\text{error}_{\text{global}} \geq \text{TOL}_{\text{GLOBAL}}$ do
           Iterate adaptive algorithm
```

```
3:     end while
4: end procedure
```

This is the method that will be used as, although the actual error will often be much lower than the error reported, a global tolerance can be guaranteed which means that the algorithm can guarantee a set degree of precision.

13.3 Adaptivity algorithms

The ability to have a non-uniform system as discussed in Section 11 means that an adaptive algorithm can be implemented.

The idea behind this is that, for a given starting implementation, the algorithm can be analysed and refined to improve the model in the areas where it is furthest from the true solution.

Therefore, on repeated iterations, a highly accurate approximation should be produced which can locally adapt to any function even if the behaviour is wildly variant over the domain as a whole.

There are two adaptations which can be made to improve the model at a given iteration: h refinement and p refinement. They work in the following way:

13.3.1 h-refinement

h refinement refers to refining the interval width $'h_k'$.

This is achieved using the following algorithm:

```
1: procedure H-REFINEMENT ON AN ELEMENT
   Let $\underline{x}$ and $\underline{p}$ be the vectors of nodal points and orders at elements of the mesh respectively.
   For element $k = [x_k, x_{k+1}]$:
2:     Compute a new node $x_{\text{mid}} = \frac{x_k + x_{k+1}}{2}$
3:     Append this node into the vector of nodes between $x_k$ and $x_{k+1}$:
           $\underline{x} = [\cdots, x_k, x_{\text{mid}}, x_{k+1}, \cdots]$
4:     Append a new value to the set of element orders such that the two elements created
   have the same order as the element they were created from
```

```
        $\underline{p} = [\cdots, p_k, p_k, p_{k+1}, \cdots]$
5:      return $\underline{x}, \underline{p}$
6:  end procedure
```

So the adaptation might look something like:

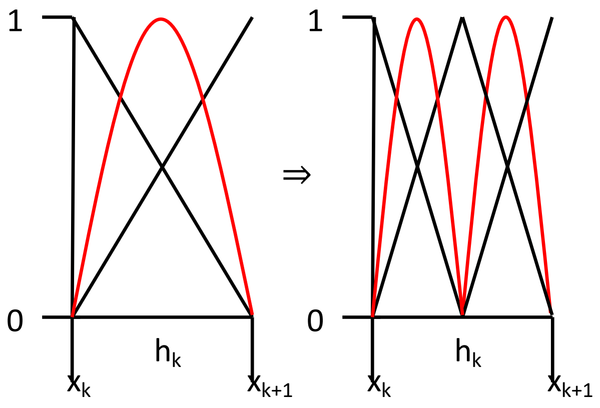

Figure 14: h refinement example

13.3.2 p-refinement

The obvious alternative refinement is the p-refinement where the p stands for the order at the element

This algorithm is very simple. The process looks like:

```
1:  procedure P-REFINEMENT ON AN ELEMENT
        Let $\underline{x}$ and $\underline{p}$ be the vectors of nodal points and orders at elements of the mesh
    respectively.
    For element $k = [x_k, x_{k+1}]$:
2:      Increase the order of this element by 1:
        $p_k = p_k + 1$
3:      return $\underline{x}$,$\underline{p}$
4:  end procedure
```

Which might look something like:

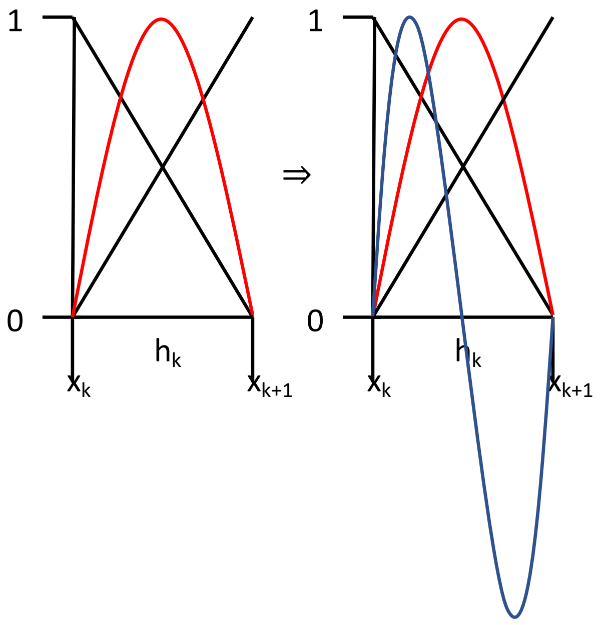

Figure 15: p refinement example

13.3.3 hp adaptive algorithm

With these tools in mind, the following algorithm can be employed to improve the model iteratively:

```
1: procedure HP ADAPTIVE ALGORITHM
2:     while error_global $\geq$ TOL_GLOBAL do
3:         Iterate the finite element method on a mesh of nodal points $\underline{x}$ and element orders $\underline{p}$
4:         Calculate an estimation of the current error on each of the intervals as some vector $\underline{E}$ and flag the worst errors as directed in (13.1).
5:         Perform an h-refinement (section 13.3.1) on a copy of the flagged elements and run the approximation again on these elements, calculating the new corresponding error estimates as a vector $\underline{E}_h$.
6:         Perform a p-refinement (section 13.3.2) on a copy of the original model's flagged elements and re-evaluate their error estimates as a vector $\underline{E}_p$ by re-running the algorithm on these elements.
7:         For each flagged element k = $[x_k, x_{k+1}]$
8:         if $(\underline{E}_h)_k \leq (\underline{E}_p)_k$ and $(\underline{E}_h)_k \leq (\underline{E})_k$ then
```

```
           Perform a p-refinement to the actual vectors $\underline{x}$ and $\underline{p}$
 9:          else if $(\underline{E}_p)_k \leq (\underline{E}_h)_k$ and $(\underline{E}_p)_k \leq (\underline{E})_k$ then
           Perform an h-refinement on the actual vectors $\underline{x}$ and $\underline{p}$
10:          end if
11:      end while
12:  end procedure
```

14 Applying an adaptive finite element method

Now all the tools required to implement the algorithm described in section 13.3.3 have been derived using the error estimate in (12.10) and the stopping criteria from section 13.2.

14.1 Demonstration of the algorithm

Applying the full algorithm to the same $tanh(100x)$ problem as used previously using the code from appendix B yields the following results:

Starting with a linear mesh of 5 elements as in 4 and demanding a global error of below 10^{-7}, the following progression can be made:

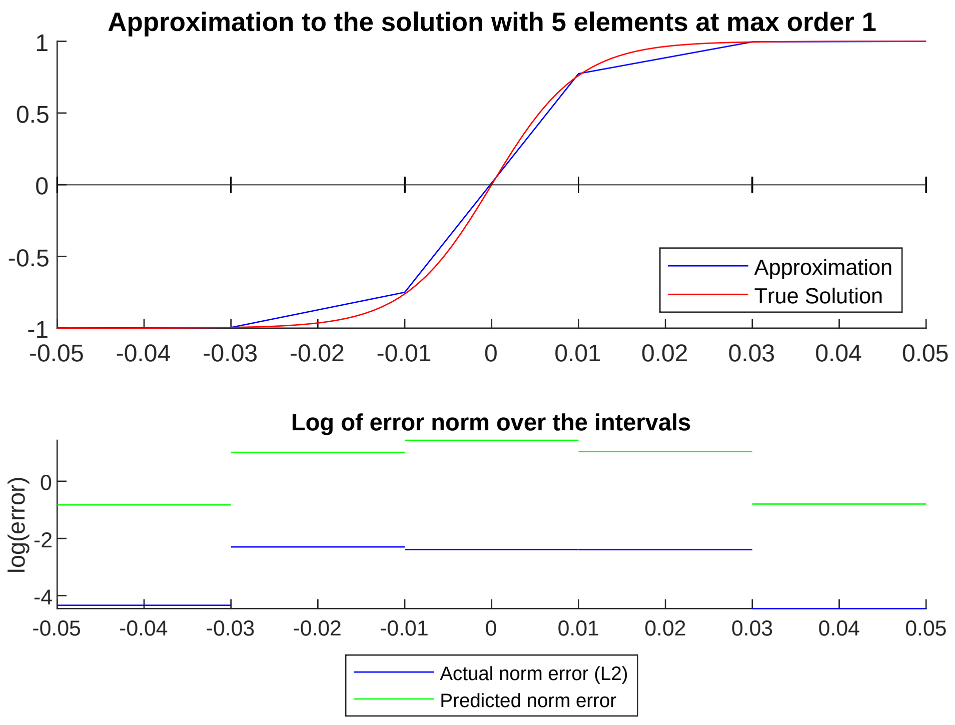

Figure 16: Initial iteration

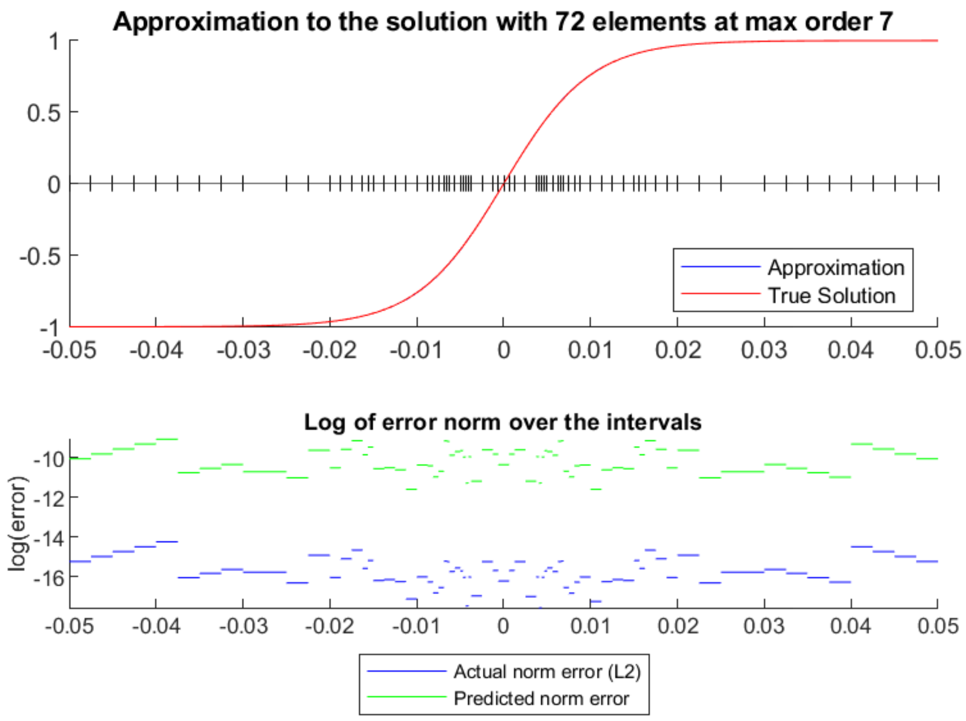

Figure 17: Final iteration

This shows the algorithm working exactly as intended with priority being given to the central slope which is the hardest to approximate and less of the computing power being devoted to improving any of the other regions.

Increasing the severity of the slope even further highlights this behaviour. Letting $k = 1000$ instead of $k = 100$ gives a final approximation:

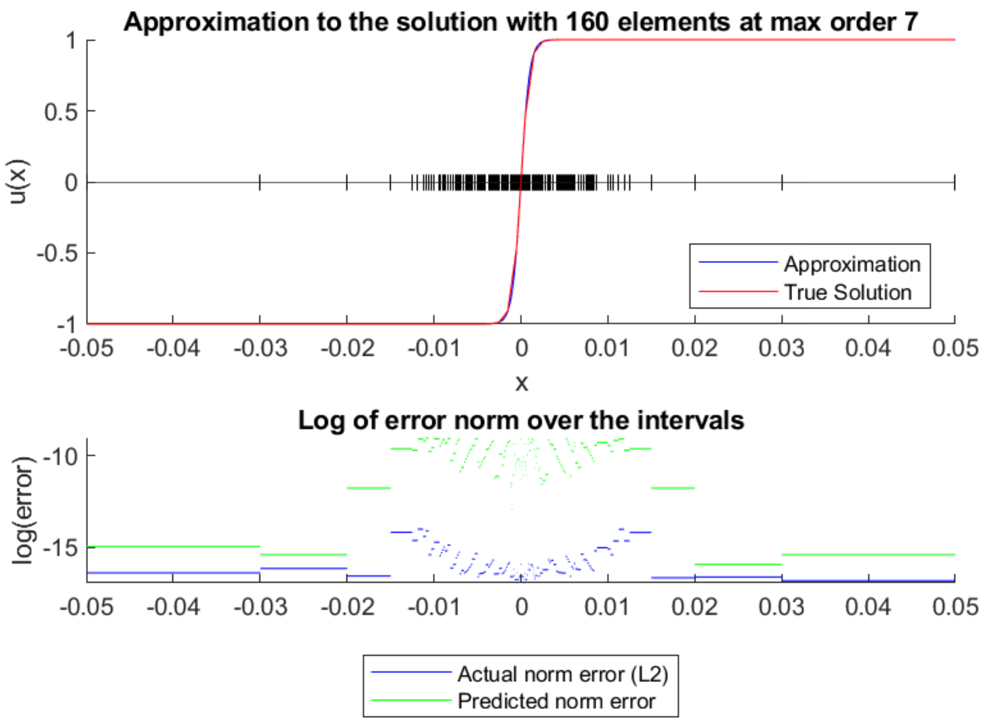

Figure 18: Final iteration (k=1000)

14.2 Demonstration of the algorithm's efficacy

14.2.1 Comparison to h or p adaptivity tanh(100x)

To see the efficacy of the full hp adaptivity, the same problem (i.e. the tanh(100x) solution function) was again approximated. This time using a h-adaptive, p-adaptive and hp adaptive method with the h-adaptive and p-adaptive just evaluated in the same way as for the hp adaptive method in 13.3.3 but where only h or p refinements were allowed respectively once the error analysis had flagged the elements with the worst errors.

Degrees of freedom analysis

Initially, a comparison of their relative degrees of freedom in the system was sought as a measure of computation required over iterations to reach certain errors. Plotting this with a linear starting mesh of 10 elements for error tolerances of 3×10^{-6},1×10^{-7} and 1×10^{-10} for h, p and hp adaptivity respectively (chosen by trial and error to ensure tolerance was reached in a sensible time) yielded the following graph:

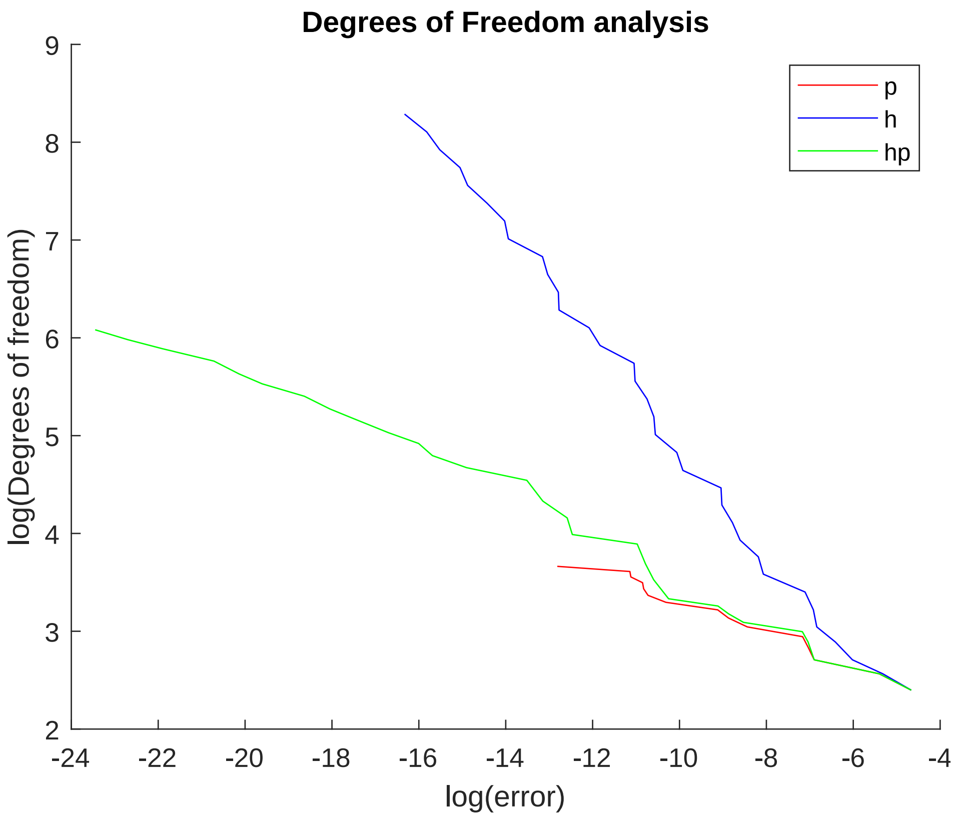

Figure 19: Error of methods against degrees of freedom required to reach it (k=100)

As expected, greater accuracy requires more degrees of freedom for all methods but their relative patterns seem to imply that, for this problem at least, p adaptivity is the most efficient method.

This may appear true from the region shown but what is omitted is that the minimum error plotted for the p adaptivity is the minimum possible error this method can reach.

This is due to the Runge phenomenon 10.1 and the algorithm's attempt to correct for it. Part of the algorithm in 13.3.3 was a condition that the refinement had to make an improvement to the error or it wouldn't be carried out.

Therefore, for a given starting mesh, each element could only be adapted to so high an order before the Runge phenomenon meant that further refinement only worsened the approximation. This means the algorithm keeps iterating and using computational power without the error decreasing at all.

The other methods were not bounded in the same way with the results implying that they can reach whatever tolerance is posed assuming the program were left running long enough.

Computation time analysis

To better illustrate the limitations of each model, an alternate approach was sought.

Instead of degrees of freedom, the time taken to process each new iteration was taken as a measure of the computing power incurred by the methods.

The algorithm was now run for a set number of refinements (in this case 25) instead of to a given tolerance. The times were then plotted against the errors reached and the following graph was produced:

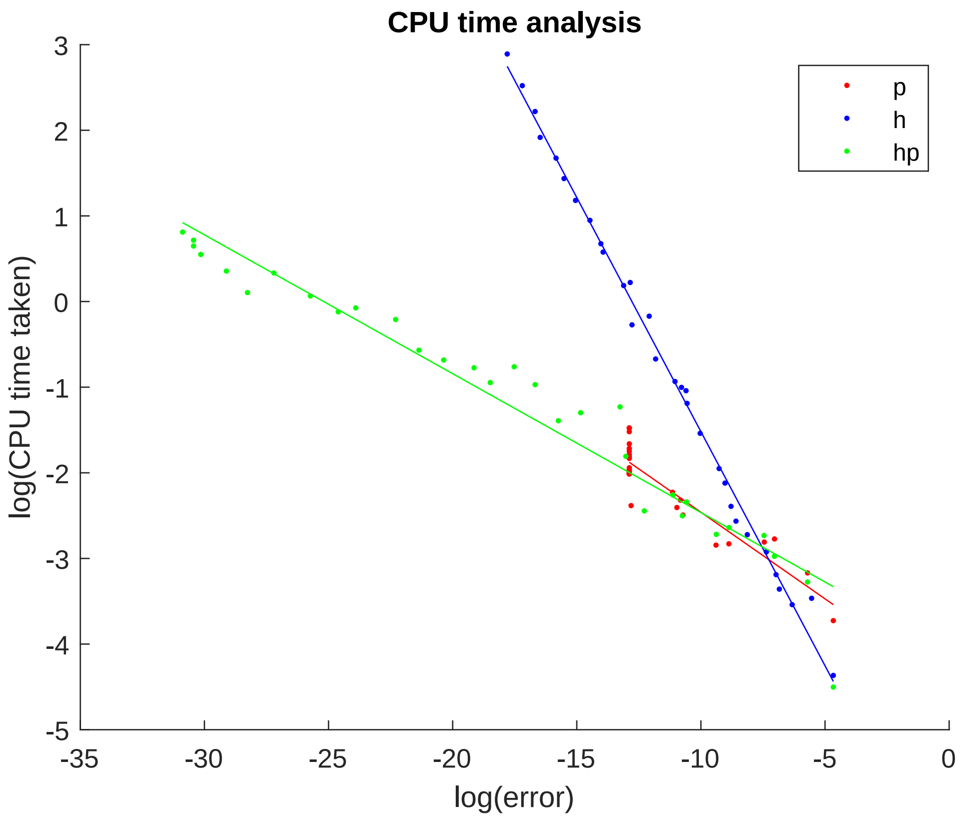

Figure 20: Error of methods against computing power required to reach it (k=100)

The same patterns are observed as before for h and hp adaptivity but now it can clearly be seen that the p adaptivity gets to a certain point and, although more iterations are being run, incurring additional computational power, error doesn't decrease at all.

This confirms what was expected, i.e. being able to adapt in both ways results in a more powerful and efficient algorithm to get an arbitrarily accurate model.

14.2.2 Comparison to h or p adaptivity tanh(1000x)

It is important to note that the relative efficacy of the three method is problem dependent. If the problem is changed to have a steeper slope in the solution as in figure 18 by setting the solution to $tanh(1000x)$ instead of $tanh(100x)$, the following pattern is observed:

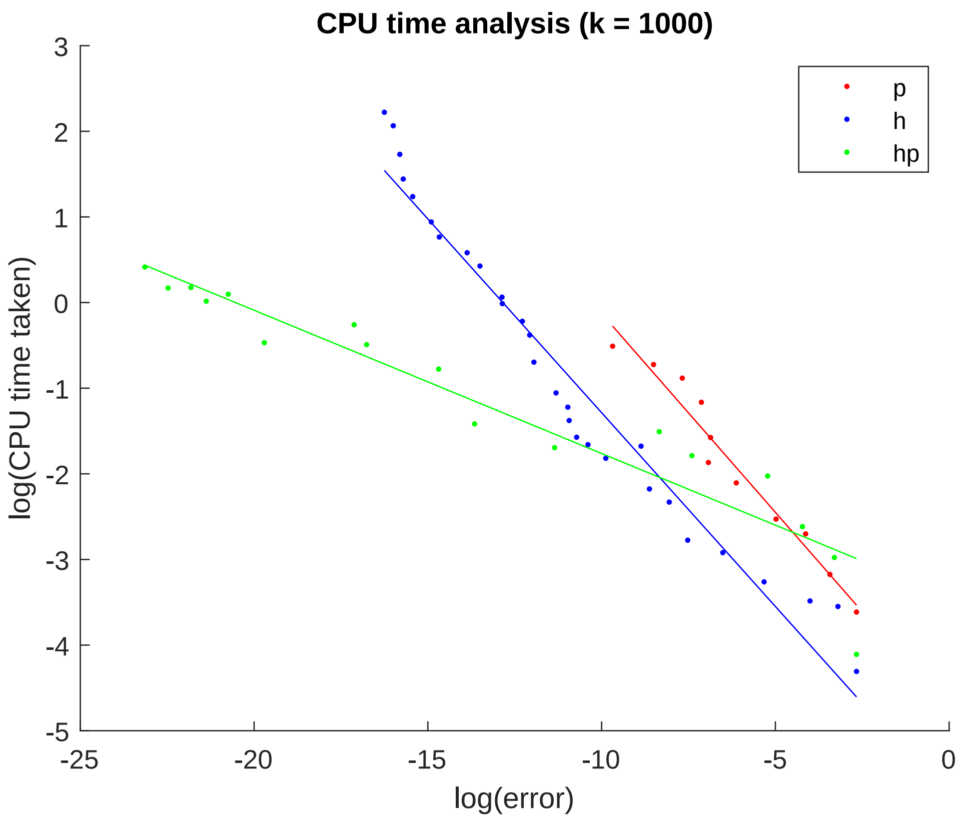

Figure 21: Error of methods against computing power required to reach it (k=1000)

Clearly, this demonstrates that the hp adaptivity method is again more efficient than either of the other individual methods.

What is also worth noting is that the h refinement is now more efficient than p refinement. This is in direct contrast to the results for k=100.

This therefore highlights that, for solutions where p refinement is more efficient and solutions where h refinement is more efficient, the hp refinement is still more efficient than either approach.

14.2.3 Comparison to global refinement

Now the same tool can be used to make the comparison between a local and simple global refinement method where each element on every iteration is refined in h and p

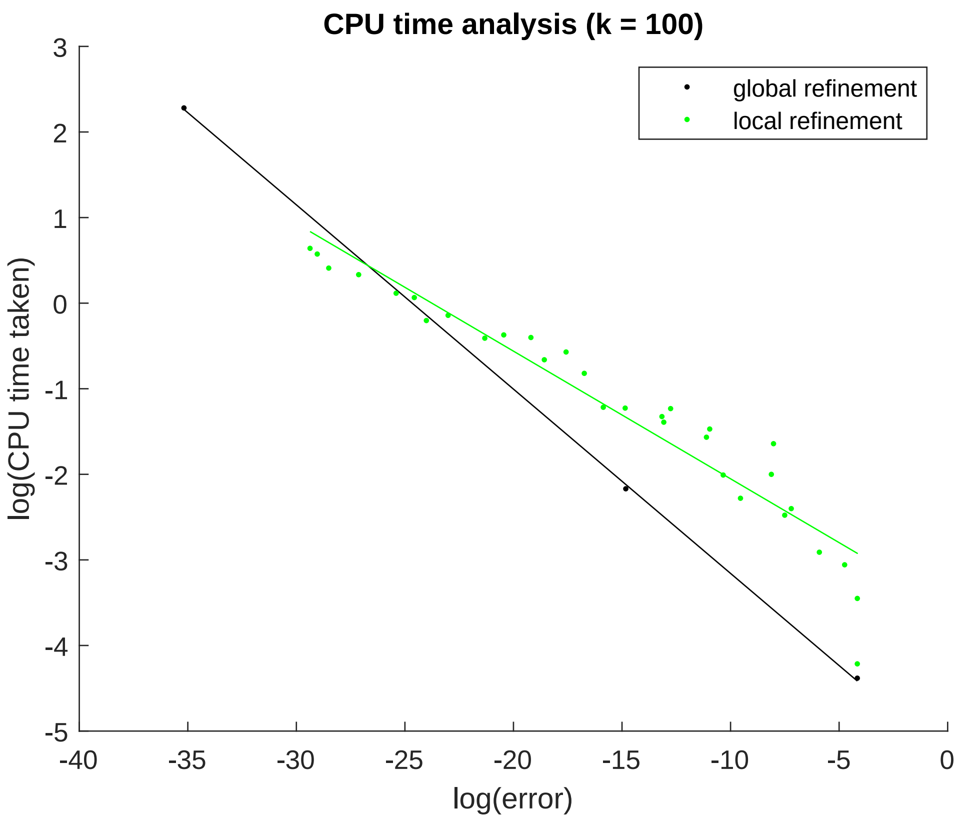

Figure 22: Error of methods against computing power required to reach it

In this case, it takes the model to be be fairly high precision before the local refinement becomes the more efficient option.

This is due to the computation required at the refinement stage. There are several evaluations of the finite element method required to evaluate whether an h or p refinement is preferable. For relatively smooth functions, the cost of these computations is greater than the computation saved by smarter refining initially. As the graph shows, the local refinement still starts to become more efficient towards the end.

To show the power of the local refinement scheme better, the step function is made steeper by changing k to 1000. Now the graph looks like:

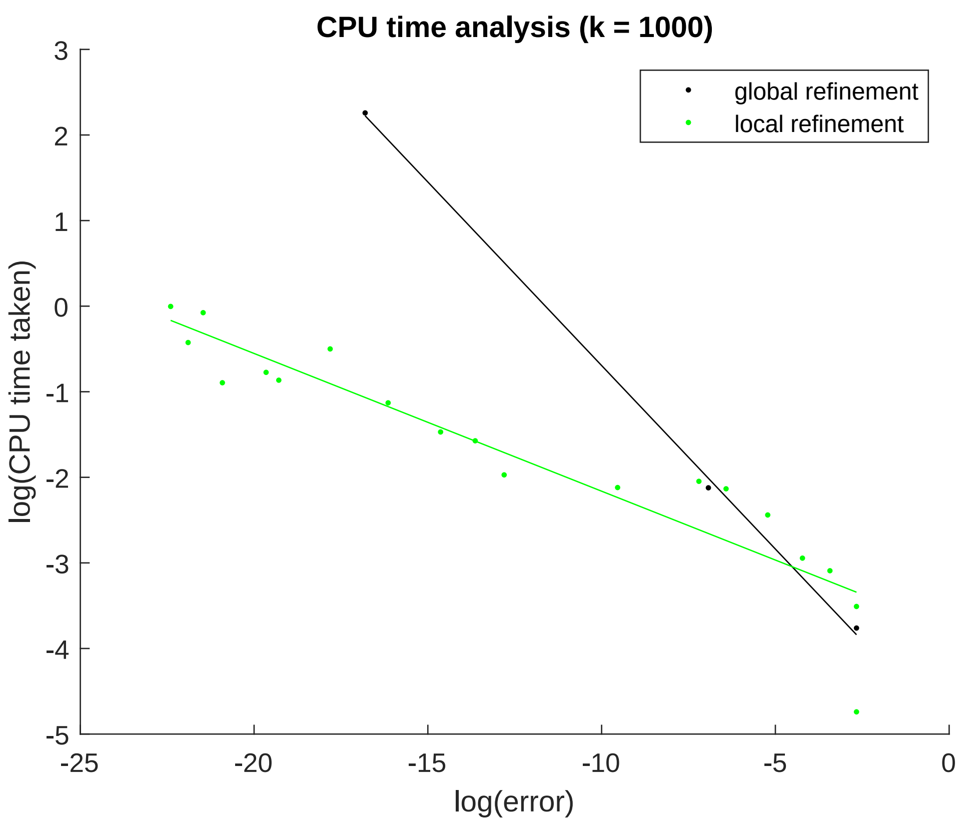

Figure 23: Error of methods against computing power required to reach it

Clearly, in this more extreme case, local refinement is far more efficient.

14.2.4 Summary

As has been shown by the analysis in 14.2.1, the hp adaptive method can be much more effective at solving systems as compared individual h or p adaptive methods.

The h methods are much more computationally expensive for the same error reduction and the p methods, although they can be as efficient, have a cap on the accuracy they can reach before the Runge phenomenons (10.1) restrict the model from improving any further.

The results in 14.2.3 also imply that, for suitably badly behaved functions, the local refinement scheme is more efficient than a global refinement scheme as the accuracy of the model required gets higher.

This implies that, for the current implementation of the method, if a lower accuracy model on a smooth function is sought, a global refinement may be preferable.

However, if the behaviour of the system isn't known and a reasonable degree of accuracy is sought, the results imply that the hp adaptive finite element method is still the most effective way to ensure that this is the case.

15 Conclusions

15.1 Summary of results

Through the work done in sections 2 to 4 , it was demonstrated that converting the model problem 7.15 to its weak form gives a well posed problem to be solved by the finite element method. Applying the finite element method to this system in a given subspace of the sobolev space $H^1_{D(A,B)}$ (2.9) was then shown in sections 5 to 6 to give a sensible approximation to the true solution of the problem.

The finite element method was then successfully applied to a standard problem at first, second and arbitrarily high order in sections 7,8 and 9 respectively by building up a suitable hierarchical construction. The validity of this model was confirmed in section 10 by comparing convergence rates of the models to the known theory. It was further illustrated in (5.4) that this gave the prescription for a model where the mesh and element orders could be whatever values were needed: giving the groundwork for an adaptive algorithm.

Having established a valid a-posteriori error estimate for the elements in section 12, this algorithm could be set up in section13 and applied to some suitable problems in section 14 to analyse its performance.

It was concluded from these analyses that the hp adaptive finite element method was highly effective at resolving badly behaved functions to high precision and efficiency, particularly as compared to h adaptive, p adaptive and global refinement methods.

15.2 Limitations

15.2.1 Efficiency for smooth functions

The hp adaptive finite element method is, as stated, highly effective for even the most complicated functions so provides a rigorous solution to most systems it may have to handle.

The error analysis in the adaptive scheme requires some computing power however which means that blind global refinement, which doesn't require this analysis, can actually be more efficient when precision required is relatively low and/or the function is suitably smooth.

This could hopefully be mitigated as an issue either by improving the efficiency of the code running the finite element method or by seeking an alternative adaptivity algorithm such as the texas 3 step method or the more complex prescription outlined in [8]

15.2.2 1D problem scope

The 1D problem posed is linear and uses only dirichlet boundary conditions.

Since the boundary conditions are localised to affecting the edge elements only, it should be relatively simple to construct approximations to derivatives at the boundaries and modify the algorithm accordingly such that Neumann or Robin boundary conditions could be solved for too. This would subtly change the formulation though, so further research would have to be done to implement this rigorously.

The problem being linear also reduces the scope of problems the system can solve although it is important to note that local behaviour of non linear systems could still be modelled by linearising said problems.

15.2.3 Higher dimensional problems

The 1D system is fairly limited in terms of practical applications. Now the algorithm works for this case, an obvious next step would be to generalise this case to a 2 dimensional finite element method which could solve for surfaces or functions over time. This would allow for real world modelling of systems.

15.2.4 Alternate basis choices

An alternative basis may have improved the efficiency of the system. For example, using a basis predicated upon Legendre polynomials as in [16] could have exploited their orthonormality condition to produce a more sparse and therefore more efficient system to solve.

15.2.5 More accurate error indicators

The error indicators, while accurate in pattern for adaptation, often predicted much higher errors than were actually present. A more accurate a-posteriori error estimate could be sought to give a more accurate estimation of the current error in the model. Schwab [16] for example proposes a similar model to the one used with some alterations which is shown in their results to be highly effective.

15.2.6 More rigorous coercivity proof

The proof 4.5.3 currently relies on the stipulation that $n \geq 0$. All the experimental testing demonstrates that the system works perfectly well even if this isn't the case so it should be possible to construct a proof which proves the results are valid for a more general case.

A Finite Element Method iteration code

```
function [u,x_vals,u_vals,priori_error_vect_L2,posteriori_error_vect]...
    = IterateFiniteElementMethod(nodes,orders,m,n,f,a,b,A,B...
    ,True_solution,Output,element_pts)
    %IterateFiniteElementMethod evaluates the finite element method ...
        on the
    %ODE -u''(x)+m*u'(x)+n*u(x) = f(x) (u(a)=A,u(b) = B) over the ...
        elements
    %with boundaries 'nodes' at set of orders 'orders'.
    %[u,x_vals,u_vals,priori_error_vect_L2,posteriori_error_vect]
    % = IterateFiniteElementMethod(nodes,orders,m,n,f,a,b,A,B
    % ,True_solution,Output,element_pts)
    %u is the vector of coefficients for the basis functions to build up
    %the full approximation where 'element_pts' points are evaluated on
    %each interval to give the points u_vals corresponding to the points
    %'x_pts' on x.
    %The approximation and real_error and posteriori_error are the ...
        vectors of real and approximated errors respectively of this
    %approximation over the intervals.
    %True solution is the correct solution to the PDE, used as a ...
        comparator
    %in the graphs. Output controls which graphs are outputted:
    % Output = 3 -> Approximation-real solution comparison graph with ...
        bases
    % and errors shown as well as matrix and vector of the system set up
    % Output = 2 -> Approximation-real solution comparison graph with ...
        bases and errors shown
    % Output = 1 -> Approximation-real solution comparison graph
    % Output = 0 -> No graphs outputted



    %% Calculating the set of bases on the interval [0,1] for the ...
        current order
    basis = cell(1,3); %Establishes the space for the bases (cells ...
        allows annonymous function entries)

    basis{1,1} = @(x) 1-x; %Linear downslope
    basis{2,1} = @(x) -1*ones(size(x)); %Derivative of the linear ...
        downslope
    basis{3,1} = @(x) zeros(size(x)); %Second derivative of the ...
        linear downslope

    basis{1,2} = @(x) x; %Linear upslope
    basis{2,2} = @(x) ones(size(x)); %Derivative of the linear upslope
    basis{3,2} = @(x) zeros(size(x)); %Second derivative of the linea ...
        upslope

    max_order = max(orders);

    N = length(nodes)-1; %This is the number of elements/intervals ...
        between the points we're evaluating

    %% General basis functions 0th derivative
    if max_order ≠1
        for pow = 2:(max_order)
            %Constructing the functions of higher order than linear
            funct = @(x) (-1)^(pow+1);
            for k = 0:(pow-1)
                funct = @(x) funct(x).*(x-k/(pow-1));
            end

            %Appending them to the basis matrix
            basis{1,pow+1} = @(x) funct(x); %This is the nth order ...
                basis function
        end
    end
```

```

%% General basis functions 1st derivative
if max_order ≠1
    for pow = 2:(max_order)
        %Constructing the functions of higher order than linear
        funct = @(x) zeros(size(x));
        for j = 0:(pow-1)
            funct_temp = @(x) (-1)^(pow+1)*ones(size(x));
            for k = 0:(pow-1)
                if k ≠j
                    funct_temp = @(x) funct_temp(x).*(x-k/(pow-1));
                end
            end
            funct = @(x) funct(x)+funct_temp(x);
        end

        %Appending them to the basis matrix
        basis{2,pow+1} = @(x) funct(x); %This is the nth order ...
            basis function 1st derivative
    end
end

%% General basis functions 2nd derivative
if max_order ≠1
    for pow = 2:(max_order)
         %Constructing the functions of higher order than linear
        funct = @(x) zeros(size(x));
        for j = 0:(pow-1) %This runs through each term in the ...
            basis and omitts this in an evaluation
            for k = 0:(pow-1)
                funct_temp = @(x) (-1)^(pow+1)*ones(size(x));
                if k ≠j
                    for l = 0:(pow-1)
                        if l ≠ k && l≠j
```

```
                                    funct_temp = @(x) ...
                                        funct_temp(x).*(x-1/(pow-1));
                               end
                          end
                          funct = @(x) funct(x)+funct_temp(x);
                     end
                end
           end

           %Appending them to the basis matrix
           basis{3,pow+1} = @(x) funct(x); %This is the nth order ...
               basis function 2nd derivative
      end
  end

  %% Calculating the set of integrals of the bases and their ...
      derivatives on the interval [0,1]
  integral_matrix = zeros(max_order+1,max_order+1,3); %Establishes ...
      a space for the integrals of these basis functions

  for k = 1:(max_order+1)
      for j = 1:(max_order+1)
           integral_matrix(k,j,1) = integral(@(x) ...
               (basis{1,k}(x)).*(basis{1,j}(x)),0,1); %This is the ...
               integral of 0th derivatives of ith and jth basis ...
               functions
           integral_matrix(k,j,2) = integral(@(x) ...
               (basis{1,k}(x)).*(basis{2,j}(x)),0,1); %This is the ...
               integral of ith and (1st derivatives of jth) basis ...
               functions
           integral_matrix(k,j,3) = integral(@(x) ...
               (basis{2,k}(x)).*(basis{2,j}(x)),0,1); %This is the ...
               integral of 1st derivatives of (ith and jth basis ...
               functions)
      end
```

```
    end

    %% Calculating set of widths of elements 'h'
    h = zeros(1,N); %Set of gaps between points
    for k = 1:N
        h(k) = nodes(k+1)-nodes(k);
    end

    %% Storing the positions of all the required orders
    positions = zeros(max_order,N); %The ith row contains the ...
        ascending number of the elements with ith order contributions.

    count = 2;

    for i = 1:max_order
        for j = 1:N
            if orders(j)≥i
                if j == N && i==1
                else
                    positions(i,j) = count;
                    count=count+1;
                end

            end
        end
    end

    %% Initialising the matrix
    non_zero_max = ...
        ((N-1)+2*(N-2)+2)+(2*2*(max_order-1)*(N-1))+((max_order-1)^2*N);
    %This is the maximum number of non-zero elements in the sparse matrix
    % ((linear-linear interactions)+(linear-higher order ...
        interactions)+(higher order-higher order interactions)

    rows = zeros(non_zero_max,1); %This is the set of the row ...
        positions of elements in the matrix
    columns = zeros(1,non_zero_max); %This is the set of the column ...
        positions of elements in the matrix
    values = zeros(1,non_zero_max); %This is the set of values in the ...
        matrix (where a((rows(i)),(columns(i)) = values(i))

    row_total = count;

    %% Appending the equation u_h(a) = A
    rows(1) = 1;
    columns(1) = 1;
    values(1) = 1;

    start = 2; %We set up 'start' to be the position in rows,columns ...
        and values after the last element appended
    %% Appending the contributions of basis functions 1-Order*N

    for k = 1:N %This iterates over the N intervals in the system
        Order = orders(k);
        %% Linear down-down interactions (k,k)
        %This appends all the interaction of the linear downslope ...
            with itself on interval k
        if  k≠1 %There's no downslope for the first interval

            %The last thing to be put in the loop is the down-down ...
                interaction in position
            % (k+1,k+1) for interval k-1.
            % This overlaps with (k,k) for interval k so we recall ...
                the same element with start

            values(start) = values(start) + ...
                1/h(k)*integral_matrix(1,1,3)...
                 +m*integral_matrix(1,1,2)+n*h(k)*integral_matrix(1,1,1);
            %This is the integral from the interaction on the kth ...
                interval of the two downs of theta(k)

            %(The integral is rescaled from the kth interval to ...
                interval [0,1]
            % so we need only refer to the bases we set up initially)
            start = start+1;

        end

        %% Linear up-down interactions (k,k+1)
        %This appends all the interaction of the linear downslope ...
            with linear upslope on interval k
        if  k≠1 && k≠N %There's no upslope for the last interval or ...
            downslope for the first interval

            rows(start) = k;
            columns(start) = k+1;
            values(start) = (1/h(k)*integral_matrix(1,2,3)...
                +m*(integral_matrix(1,2,2))...
                +n*(h(k)*integral_matrix(1,2,1)));
            %This is the integral from the interaction on the kth ...
                interval
            % of the down of theta(k) and the up of theta(k+1)

            rows(start+1) = k+1;
            columns(start+1) = k;
            values(start+1) = (1/h(k)*integral_matrix(2,1,3)...
                +m*(integral_matrix(2,1,2))...
                +n*(h(k)*integral_matrix(2,1,1)));
            %This is the integral from the interaction on the kth ...
                interval
            % of the up of theta(k+1) and the down on theta(k)

            start = start+2;
```

```

    end

    %% Linear- higher order interactions (k+1,k+(j-1)N)
    if Order>1
        for j = 2:(Order)
            %Down-Higher order interactions (k+1, - )
            %This appends all the higher order interactions of ...
                the linear downslope on the kth interval
            if k≠N
                rows(start) = k+1;
                columns(start) = positions(j,k);
                values(start) = (1/h(k)*integral_matrix(2,j+1,3)...
                    +m*(integral_matrix(2,j+1,2))...
                    +n*(h(k)*integral_matrix(2,j+1,1)));
                %This is the integral from the interaction on the ...
                    kth interval
                % of the down of theta(k) and the order j basis ...
                    function
                % (in position j+1 as there are 2 linear functions)

                rows(start+1) = positions(j,k);
                columns(start+1) = k+1;
                values(start+1) = (1/h(k)*integral_matrix(j+1,2,3)...
                    +m*(integral_matrix(j+1,2,2))...
                    +n*(h(k)*integral_matrix(j+1,2,1)));
                %This is the integral from the interaction on the ...
                    kth interval
                % of the down of theta(k) and the order j basis ...
                    function

                start = start+2;
            end

            %Up-Higher order interactions (k, - )
```

```
                %This appends all the higher order interactions of ...
                    the linear upslope on the kth interval
                if k≠1
                    rows(start) = k;
                    columns(start) = positions(j,k);
                    values(start) = 1/h(k)*integral_matrix(1,j+1,3)...
                        +m*(integral_matrix(1,j+1,2))...
                        +n*(h(k)*integral_matrix(1,j+1,1));
                    %This is the integral from the interaction on the ...
                        kth interval
                    % of the up of theta(k) and the order j basis ...
                        function

                    rows(start+1) = positions(j,k);
                    columns(start+1) = k;
                    values(start+1) = 1/h(k)*integral_matrix(j+1,1,3)...
                        +m*(integral_matrix(j+1,1,2))...
                        +n*(h(k)*integral_matrix(j+1,1,1));
                    %This is the integral from the interaction on the ...
                        kth interval
                    % of the down of theta(k) and the order j basis ...
                        function

                    start = start+2;
                end
            end
        end

        %% Higher order-higher order interactions

        for Order_row = 2:Order
            for Order_col = 2:Order
                rows(start) = positions(Order_row,k);
                columns(start) = positions(Order_col,k);
                values(start) = ...
                    1/h(k)*integral_matrix(Order_row+1,Order_col+1,3)...
                    +m*(integral_matrix(Order_row+1,Order_col+1,2)) ...
                    +n*(h(k)*integral_matrix(Order_row+1,Order_col+1,1));
                                 %This is the integral from the ...
                                     interaction on the kth interal
                                 % of the Order_row and Order_col ...
                                     basis functions

                start = start+1;
            end
        end

        %% Linear down-down interactions (k+1,k+1)
        if k≠N
            rows(start) = k+1;
            columns(start) = k+1;
            values(start) = 1/h(k)*integral_matrix(2,2,3)...
                +m*(integral_matrix(2,2,2))...
                +n*(h(k)*integral_matrix(2,2,1));
            %This is the integral from the interaction on the kth ...
                interval
            % of the two linear downs of theta(k)
        end

    end

    %% Appending the equation u_h(b) = B
    rows(start) = row_total;
    columns(start) = row_total;
    values(start) = 1;

    rows = rows(1:start);
    columns = columns(1:start);
    values = values(1:start);
```

```

%% Generating the matrix
S = sparse(rows,columns,values,row_total,row_total);

if Output == 3
    fprintf("Matrix = \n")
    disp(full(S))
end
%% Computing the corresponding vector

F = zeros(row_total,1);

F(1) = A;
F(row_total) = B;

for k = 1:N %This iterates over the intervals
    Order = orders(k);
    %% Upslopes
    if k ≠ (N)
        F(k+1) = F(k+1) + h(k)*...
            integral(@(x)f(h(k)*x+nodes(k)).*(basis{1,2}(x)),0,1);
        %Upslope interactions on the kth interval (theta(k+1)
    end

    %% Downslopes
    if k ≠ 1
        F(k) = F(k) + h(k)*...
            integral(@(x)f(h(k)*x+nodes(k)).*(basis{1,1}(x)),0,1);
        %Downslope interactions on the kth interval (theta(k))
    end

    %% Higher orders

    if Order ≠1
        for j = 2:(Order)
```

```
                F(positions(j,k)) = F(positions(j,k))...
                    + h(k)*integral(@(x)f(h(k)*x+nodes(k)).*...
                    (basis{1,j+1}(x)),0,1);
                %Higher order interactions on the kth interval ...
                    (theta(k+(j-1)N))
            end
        end
    end

    if N≥2
    %% Removing interactions with theta(0) (down)
    F(2) = F(2)-A*(1/h(1)*integral_matrix(1,2,3)...
        +m*(integral_matrix(2,1,2))...
        +n*(h(1)*integral_matrix(1,2,1)));
    %Removes the interaction of theta(0) (down) with theta(1) (up)

    Order = orders(1);
    if Order≠1
        for k = 1:Order-1
            F(positions(k+1,1)) = ...
                F(positions(k+1,1))-A*(1/h(1)*integral_matrix(1,k+2,3)...
                +m*(integral_matrix(k+2,1,2))...
                +n*(h(1)*integral_matrix(1,k+2,1)));
            %This removes the interaction of theta(0) (down) with any ...
                other bases
        end
    end


    %% Removing interactions on the last interval (up)
    F(N) = F(N)-B*(1/h(N)*integral_matrix(2,1,3)...
        +m*(integral_matrix(1,2,2))+n*(h(N)*integral_matrix(2,1,1)));
    %Removes the interaction of theta(end) (up) with theta(N-1) (down)

    Order = orders(N);
    if Order≠1
        for k = 1:Order-1
            F(positions(k+1,N)) = ...
                F(positions(k+1,N))-B*(1/h(N)*integral_matrix(2,k+2,3)...
                 +m*(integral_matrix(k+2,2,2))...
                 +n*(h(N)*integral_matrix(2,k+2,1)));
            %This removes the interaction of theta(N) (up) with any ...
                other bases
        end
    end
    end
    if Output == 3
        fprintf("Vector = \n")
        disp(F')
    end

    %% Solving the sysytem
    u = linsolve(full(S),F);

    if Output == 3
        fprintf("Values = \n")
        disp(u')
    end

    %% Evaluating modelled points and true error
    %element_pts = Number of points evaluated on each interval
    x_vals = zeros(1,element_pts*N);
    u_vals = zeros(1,element_pts*N);

    tempppts = linspace(0,1,element_pts);

    priori_error_vect_L2 = zeros(1,N);

    for k = 1:N
        interval = (element_pts*(k-1)+1):(element_pts*k);
        x_vals(interval) = h(k)*temppts+nodes(k);
        Order = orders(k);
        interval_model = @(x) zeros(size(x));

        % Linear contributions
        if k ≠N
            interval_model = @(x) interval_model(x)...
                + u(k)*basis{1,1}(x)+u(k+1)*basis{1,2}(x);
        else
            interval_model = @(x) interval_model(x) ...
                + u(k)*basis{1,1}(x)+u(row_total)*basis{1,2}(x);
        end

        % Higher-order contributions
        if Order≠1
            for j = 1:Order-1
                interval_model = @(x) interval_model(x) ...
                    + u(positions(j+1,k)).*basis{1,j+2}(x);
            end
        end
        u_vals(interval) = interval_model(temppts);

        %L2 norm
        L2 = h(k)*(integral(@(x)(True_solution(h(k)*x+nodes(k))...
            -interval_model(x)).^2,0,1));

        priori_error_vect_L2(k) = sqrt(L2);
    end

    %% Calculating aposteriori error
    posteriori_error_vect = zeros(1,N);

    for k = 1:N
        Order = orders(k);
        res = @(x) f(h(k)*x+nodes(k));

        % Linear contributions
        if k ≠N
            res = @(x) res(x) - ...
                (u(k).*((m/h(k)).*basis{2,1}(x)+n.*basis{1,1}(x))...
                +u(k+1).*((m/h(k)).*basis{2,2}(x)+n.*basis{1,2}(x)));
        else
            res = @(x) res(x) - ...
                (u(k).*((m/h(k)).*basis{2,1}(x)+n.*basis{1,1}(x))...
                +u(row_total).*((m/h(k)).*basis{2,2}(x)+n.*basis{1,2}(x)));
        end

        % Higher-order contributions
        if Order≠1
            for j = 1:Order-1
                res = @(x) res(x)-u(positions(j+1,k)).*...
                    (-1*(1/h(k)).^2.*basis{3,j+2}(x)...
                    +(m/h(k)).*basis{2,j+2}(x)+n.*basis{1,j+2}(x));
            end
        end

        res = @(x) res(x).^2;

        if k ≠ N
            RL2 = (h(k)^2/(orders(k)*orders(k+1)))*...
                    h(k)*integral(@(x)res(x),0,1);
        else
            RL2 = (h(k)/Order)^2*h(k)*integral(@(x)res(x),0,1);
        end

        posteriori_error_vect(k) = sqrt(RL2);
    end

    %% Creating and plotting functions

    if Output≠0
        if Output == 1
            %% Plotting model
            figure
            hold on
            p1 = plot(x_vals,u_vals,'b');
            yline(0)
            plot(nodes,zeros(length(nodes)),'k|')

            %% Plotting true solution for comparison
            pts2 = linspace(a,b,100); %Set of points the true ...
                function is evaluated over
            p2 = plot(pts2,True_solution(pts2),'r');
            legend([p1 p2],'Approximation','True ...
                Solution','Location','best')
            title(sprintf("Approximation to the function with %d ...
                elements at max order %d",N,max_order))

            hold off
        else
            %% Plotting model
            figure
            subplot(2,1,1)
            hold on
            p1 = plot(x_vals,u_vals,'b');

            yline(0)
            plot(nodes,zeros(length(nodes)),'k|')

            %% Plotting true solution for comparison
            pts2 = linspace(a,b,100); %Set of points the true ...
                function is evaluated over
            p2 = plot(pts2,True_solution(pts2),'r');
            legend([p1 p2],'Approximation','True ...
                Solution','Location','best')
            title(sprintf("Approximation to the solution with %d ...
                elements at max order %d",N,max_order))
            hold off

            %% Plotting errors over the intervals
            subplot(2,1,2)
            hold on

            for i = 1:N
                interval = (element_pts*(i-1)+1):(element_pts*i);
                plot(x_vals(interval)...
                    ,log(sqrt(priori_error_vect_L2(i)))*ones(1,element_pts),'b')
                plot(x_vals(interval)...
                    ,log(sqrt(posteriori_error_vect(i)))*ones(1,element_pts),'g')
            end

            ylabel('log(error)')
            legend('Actual norm error (L2)','Predicted norm ...
                error','Location','southoutside')
            title("Log of error norm over the intervals")
            hold off
        end
    end
end
```

B Adaptive method code

```
%% Setup for -u''(x) +mu'(x)+nu(x)= f(x)

m = 10;
n = 20;

a = -0.05; %u(a) = A
b = 0.05; %u(b) = B

k=1000; %Controls steepness

f = @(x) 2*k^2*tanh(k*x).*sech(k*x).^2+m*k*(sech(k*x)).^2+n*tanh(k*x);
True_solution = @(x) tanh(k*x);

A = True_solution(a);
B = True_solution(b);

%% Initialising
iteration = 1;
N = 5;
element_pts = 100;
pts = linspace(a,b,N+1); %Set of evenly distributed xi points
starting_order = 1;
orders = starting_order*ones(1,N);

percentage_refined = 80; %The elements with top percentage_refined ...
    percent of worst errors will be refined

Global_TOL = 10^-7;
%% Starting iteration
[¬,x_vals,y_vals,priori_error_vect_L2,posteriori_error_vect] = ...
    IterateFiniteElementMethod(pts,orders,m,n,f,a,b,A,B...
    ,True_solution,2,element_pts);
```

```

global_error_L2 = norm(priori_error_vect_L2);
global_error_posteriori= norm(posteriori_error_vect);
fprintf("Iteration %d \n",iteration)
fprintf("Global error L2: %e \n",global_error_L2)
fprintf("Global predicted error: %e \n",global_error_posteriori)
fprintf("\n")


%% Iterating
while global_error_posteriori ≥ Global_TOL

    N = length(pts)-1;
    Percentile = prctile(posteriori_error_vect,percentage_refined);
    %This is the boundary for the upper 'percentage_refined' ...
        percentile of
    %the error vector

    %% Updating intervals
    e_vals_intervals=  zeros(1,N);

    i = 1;
    while i≤N
        if posteriori_error_vect(i) ≥ Percentile
            j=0;
            if i≠N
                while posteriori_error_vect(i+j+1) ≥ Percentile %This ...
                    captures any sequential refinement regions into ...
                    one interval.
                    j = j+1;
                    if i+j+1 ≥N+1 %Cuts the iteration before it tries ...
                        to access a point outside the vector
                        break
                    end
                end
            end

            temp_pts = linspace(pts(i),pts(i+j+1),2*j+3);
            %Generates and interval between the first and last node ...
                in the interval with an extra node between each fo ...
                the existing nodes

            N_temp = length(temp_pts)-1;

            temp_orders = zeros(1,N_temp);

            count = 1;

            for q = 0:j
                temp_orders(count) = orders(i+q);
                temp_orders(count+1) = orders(i+q);
                count = count+2;
            end

            [¬,¬,¬,¬,interval_errors] ...
                = ...
                    IterateFiniteElementMethod(temp_pts,temp_orders,m,n,f...
                ,pts(i),pts(i+j+1),y_vals(element_pts*(i-1)+1)...
                ,y_vals(element_pts*(i+j)), True_solution,0,element_pts);

            for k = 1:j+1
                e_vals_intervals(i) = ...
                    sqrt(interval_errors(k)^2+interval_errors(k+1)^2);
                i = i+1;
            end
        else
            e_vals_intervals(i) = posteriori_error_vect(i);
            i=i+1;
        end
    end
```

```

%% Updating orders
e_vals_orders = zeros(1,N);

i = 1;
while i≤N
    if posteriori_error_vect(i) ≥ Percentile
        j=0;
        if i≠N
            while posteriori_error_vect(i+j+1) ≥ Percentile
                j = j+1; %There are j+1 intervals in the current ...
                    run of elements to be refined.
                if i+j+1 ≥N+1
                    break
                end
            end
        end

        temp_pts = pts(i:i+j+1);

        N_temp = length(temp_pts)-1;

        temp_orders = orders(i:i+j)+ones(1,j+1);

        [¬,¬,¬,¬,order_errors] = ...
            IterateFiniteElementMethod(temp_pts,temp_orders,m,n,f...
            ,pts(i),pts(i+j+1),y_vals(element_pts*(i-1)+1)...
            ,y_vals(element_pts*(i+j)),True_solution,0,element_pts);

        for k = 1:j+1
            e_vals_orders(i) = order_errors(k);
            i = i+1;
        end
    else
        e_vals_orders(i) = posteriori_error_vect(i);
```

```
            i=i+1;
        end
    end

    %% Update
    count = 0;
    for i = 1:N
        if posteriori_error_vect(i) ≥ Percentile
            if  e_vals_intervals(i)≤ posteriori_error_vect(i)&& ...
                e_vals_intervals(i)≤e_vals_orders(i)
                new_interval = ...
                    linspace(pts(i+count),pts(i+count+1),3); ...
                    %Generates a new interval
                new_entry = new_interval(2); %Takes off the ...
                    boundaries to avoid duplicates
                pts = ...
                    [pts(1:i+count),new_entry...
                    ,pts(i+1+count:length(pts))]; %Puts the new ...
                        interval in the vector
                orders = ...
                    [orders(1:i+count),orders(i+count)...
                    ,orders(i+count+1:length(orders))]; %Updates the ...
                        list of orders for the new intervals
                count = count + 1;
            elseif e_vals_orders(i)≤ posteriori_error_vect(i) && ...
                e_vals_orders(i)≤e_vals_intervals(i)
                orders(i+count) = orders(i+count)+1;
            end
        end
    end


    %% Run new model
    [¬,x_vals,y_vals,priori_error_vect_L2,posteriori_error_vect] ...
        = IterateFiniteElementMethod(pts,orders,m,n,f,a,b,A,B...
        ,True_solution,2,element_pts);

    iteration = iteration+1;

    global_error_L2 = norm(priori_error_vect_L2);
    global_error_posteriori= norm(posteriori_error_vect);
    fprintf("Iteration %d \n",iteration)
    fprintf("Global error L2: %e \n",global_error_L2)
    fprintf("Global predicted error: %e \n",global_error_posteriori)
    fprintf("\n")

end
```

References

[1] Ainsworth, M. and Oden, J. T. (1997). A posteriori error estimation in finite element analysis. Computational Mechanics Advances.

[2] Babuška, I. and Szabó, B. (1982) 'On the rates of convergence of the finite element method', International Journal for Numerical Methods in Engineering, 18(3), pp. 323-341.

[3] Szabó, B.A. and Mehta, A.K. (1978) 'p-Convergent Finite Element Approximations in Fracture Mechanics', International Journal for Numerical Methods in Engineering.

[4] Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. Bulletin of the American Mathematical Society.

[5] Ern, A. and Guermond, J. L. (2004). Theory and Practice of Finite Elements. Springer.

[6] Ghesmati, A., Bangerth, W. and Turcksin, B. (2018). Residual-based a posteriori error estimation for hp-adaptive finite element methods for the Stokes equations. de Gruyter.

[7] Gratsch, T. and Bathe, K. J. (2004). A posteriori error estimation techniques in practical finite element analysis. Computers and structures 83(2005)235–265.

[8] Houston, P. and Suli, E. (2004). A note on the design of hp-adaptive finite element methods for elliptic partial differential equations. Computer Methods in Applied Mechanics and Engineering.

[9] Ciarlet, P. (2002). The Finite Element Method For Elliptic Problems. Classics In Applied Mathematics, 40.

[10] Johnson, C. and Szepessy, A. (1995). Adaptive Finite Element Methods for Conservation Laws Based on a Posteriori Error Estimates. Communications on Pure and Applied Mathematics.

[11] Larson, M. G. and Bengzon, F. (2013). The Finite Element Method: Theory, Implementation, and Applications. Springer.

[12] Evans, L. C. (2010). Partial Differential Equations. 2nd ed. Providence, Rhode Island: American Mathematical Society.

[13] Ern, A., and Guermond, J.L. (2004). Theory and Practice of Finite Elements. Springer.

[14] Poole, D. (2015). Linear Algebra: A Modern Introduction. Cengage Learning

[15] Repin, S. (2008). A Posteriori Estimates for Partial Differential Equations. Walter de Gruyter.

[16] Schwab, C. (1998). P and HP Finite Element Methods: Theory and Applications in Solid and Fluid Mechanics. Oxford: Clarendon Press.

[17] Schwab, C. and Suri, M. (1996). The p and hp versions of the finite element method for problems with boundary layers. Mathematics of Computation, 65.

[18] Solin, P., Segeth, K. and Dolezel, I. (2004). Higher-Order Finite Element Methods. Chapman and Hall/CRC.

[19] Strauss, W. (2008). Partial differential equations: an introduction (Vol. 4). John Wiley and Sons

[20] Chi-Wang Shu (2003): High-order Finite Difference and Finite Volume WENO Schemes and Discontinuous Galerkin Methods for CFD, International Journal of Computational Fluid Dynamics, 17:2, 107-118

[21] Süli, E. and Mayers, D. F. (2003). An Introduction to Numerical Analysis. Cambridge University Press

[22] Kouris, C., Dimakopoulos, Y., Georgiou, G. and Tsamopoulos, J. (2002). Comparison of spectral and finite element methods applied to the study of the core-annular flow in an undulating tube. International Journal for Numerical Methods in Fluids

[23] Sauer, T. (2012). Numerical Analysis. Pearson Education.