School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Learning Resources for Understanding and Using the Finite Element Method

Kyle Antony Morris

Submitted in accordance with the requirements for the degree of Computer Science with Mathematics

2022/23

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (DD/MM/YY)
<Example> Scanned participant consent forms	PDF file / file archive	Uploaded to Minerva (DD/MM/YY)
<Example> Link to online code repository	URL	Sent to supervisor and assessor (DD/MM/YY)
<Example> User manuals	PDF file	Sent to client and supervisor (DD/MM/YY)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) ______________________

Summary

The aim of this project is to address the issue of the mathematical complexity of the finite element method and create a set of learning resources that can be used to bridge the gap in understanding required to use FEM. The goal is to guide students through the mathematics as well as instruct them on how to implement the method themselves. In this report I will describe the background research done to influence my design decisions, detail the writing process, and discuss the results obtained from user feedback.
COME BACK TO THIS

Acknowledgements

<The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.>

Note that it is not acceptable to solicit assistance on 'proof reading' which is defined as the "the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test"; see
`https://www.leeds.ac.uk/secretariat/documents/proof_reading_policy.pdf`

Contents

Chapter 1

Introduction and Background Research

1.1 Introduction

Partial differential equations (PDEs) are some of the most important types of formulae in mathematics. They arise in every field of mathematically inclined science, such as physics, engineering, chemistry and more. PDEs are used to describe physical systems. Some PDEs describe dynamical systems; that is, systems that change with time. Others describe static systems, that only vary in space, or sometimes other quantities. This might include the movement of fluids, heat and waves, or the deformation of certain structures acting under a force. They are used to model population dynamics, chemical reactions, electromagnetic fields and are also central to quantum mechanics. PDEs are fundamental to our understanding of science and the world[1].

Since PDEs are so important, the natural question is, how are they solved? Solving a PDE simply means obtaining an equation that describes the key variable explicitly, e.g. velocity of fluid particles given their coordinates in space, number of individuals in a population at any time, or amount of heat at a certain point on a metal rod. There are methods for solving certain types of PDE analytically, but generally PDEs are either incredibly difficult to solve, or in many cases, impossible.

Fortunately, methods have been developed to provide approximate solutions to PDEs, called numerical methods. These methods always have some degree of error, but by increasing the amount of computation performed, this error can be reduced. Numerical methods can involve an enormous amount of computation, making them the ideal candidates for modern computing, where processors perform billions of operations per second.

One such method is called the Finite Element Method (FEM), where the domain being solved over is discretised, and linear algebra techniques are used to construct a linear system of equations that can be numerically solved to obtain our approximate solution[2]. The method has a strong mathematical foundation and it can be shown that the approximation does converge as the number of discrete elements increases[3]. The method is fantastically versatile, as it can be applied to any PDE, whether linear or non-linear, the only problem is that it requires a good mathematical understanding to implement.

This finite element method has its origins in a numerical algorithm called the Rayleigh-Ritz method from the late nineteenth/early twentieth century, which was used for predicting stress and displacement on solid structures[4]. Later an alternative to this was developed called the Galerkin method, equivalence between these two was later proven in 1962 [5]. In 1956, the first academic paper was published which coined the term 'finite element' by Turner et al. [6], the main advantage of this over the previous methods was its application to complex domains. After this the use of the method began to spread and it was mathematically proven in a government report in 1972 [7]. Since then, the method has been ubiquitous throughout physics, engineering and science as a whole.

1.2 The Finite Element Method

1.2.1 How the Finite Element Method Works

As previously mentioned, FEM is a numerical method for solving differential equations. Most often, it is applied to partial differential equations. The general form of a PDE is shown in Equation 1.1. Here, the variables are denoted $t, x_1, \cdots, x_n$, of which there are $n+1$. The order of the equation is the order of the highest derivative of u, denoted m. u is the unknown function that being solved for. Note that t may not always be present here.

$$\begin{aligned} F\Big(&u, t, x_1, x_2, \cdots, x_n \\ &\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \cdots, \frac{\partial u}{\partial x_n}, \\ &\frac{\partial^2 u}{\partial^2 x_1^2}, \frac{\partial^2 u}{\partial x_2^2}, \cdots, \frac{\partial^2 u}{\partial x_n^2}, \\ &\qquad\cdots \\ &\frac{\partial^m u}{\partial x_1^m}, \frac{\partial^m u}{\partial x_2^m}, \cdots, \frac{\partial^m u}{\partial x_n^m}\Big) = 0 \end{aligned} \tag{1.1}$$

Clearly, Equation 1.1 is a complicated equation, but most PDEs do not have this many variables. Also, it is rare that a PDE will have an order greater than two, although this can occur. Along side the equation itself, there will also be a set of conditions that constrain the variables to some domain Ω. There will also be a set of conditions that constrain u on the boundary of this domain $\delta\Omega$. These are called the boundary conditions. If one of the variables involved is time this must also be constrained by an initial condition. Together these make a initial value boundary problem (IVBP). A full instance can be seen in Equation 1.2.

$$\begin{cases} F = 0 & \text{on } \Omega \\ u = u_D(x_1, \cdots, x_n, t) & \text{on } \delta\Omega \\ u = u_{\text{init}}(x_1, \cdots, x_n) & \text{at } t = 0 \end{cases} \tag{1.2}$$

$$\text{where } \Omega \subset \mathbb{C}^n$$

The method of solving such an equation using the finite element method first starts by converting Equation 1.2 into a variational formulation. Specific methods of constructing this formulation vary from problem to problem but the general approach is as follows:

- Take the equation defined over Ω and multiply both sides by a test function v.
- Integrate both sides over Ω.
- Integrate any second order derivatives by parts.
- Rearrange so all terms involving u are on one side. This expression is called the bilinear form, denoted $a(u, v)$ and the other side is called the linear form, denoted $L(v)$.

After doing this, a few facts about the functions spaces u and v belong to need to be acknowledged. The class of functions that can solve Equation 1.2 belong to the function space,

$$V = \{v : v \in H^1(\Omega), v = u_D \text{ on } \delta\Omega, v = u_{\text{init}} \text{ at } t = 0\}$$

Clearly, in addition to this, they must also satisfy the main equation. $H_1(\Omega)$ is the Sobloev space, which is a space where all functions must be square integrable and continuous and their derivatives must simply be square integrable. V is called the trial space, and u is called the trial function. In order for the variational formulation to be valid, the test function must also come from $H_1(\Omega)$, but instead include the condition that it must vanish on the boundary of our domain. This yields the test space,

$$V_0 = \{v : v \in H^1(\Omega), v = 0 \text{ on } \delta\Omega, \}$$

This trick allows us to remove any parts of our integration that include v on $\delta\Omega$. More importantly however, subspaces of V and V_0, denoted V_h and $V_{h,0}$ can be constructed, where the elements are piecewise polynomial functions. This can only be done because these spaces permit functions with discontinuous derivatives. So the transition from a continuous space to a discontinuous one is made, replacing u with $u_h \in V_0$ and assuming $v \in V_{h,0}$. This is the finite element approximated problem.

$$\begin{aligned}&\text{Solve for } u_h \in V_h \text{ where,}\\ &a(u_h, v) = L(v) \quad \forall v \in V_{h,0}\end{aligned} \tag{1.3}$$

To solve this problem, our domain Ω must be split into discrete segments, called a mesh. In one dimension, this is amounts to splitting the number line into sub-intervals, in two dimensions, the plane is split into cells, or finite elements. These cells are primitive shapes like rectangles or triangles and depending on the domain, either it can be simpler to use one or the other. The points where these cells meet each other are called nodes.
Now to solve, techniques from linear algebra are used. A basis of V_h and $V_{h,0}$ must be constructed to rewrite u and v as a linear combination of basis functions. The one used in the finite element method is called the nodal basis, because the coefficients of a linear combination of each of these basis vectors are equal to the function at the nodes of the mesh. The usefulness of this, is that any function can be defined using only these nodal values, meaning if they can be obtained, the approximate solution can be found.
The way to obtain these coefficients, called degrees of freedom, is by substituting in u and v as a linear combination of these basis functions, called hat functions. This produces $n + 1$ independent equations, where n is the number of nodes in the mesh. Then, the resulting system of equations can be solved using known numerical methods. One can use direct methods like Gaussian elimination and LU-Factorisation, or where appropriate, iterative methods like Jacobi, or Gauss-Seidel.
If time is a variable in the equation, one can use the numerical methods used for ordinary differential equations to discretise the time domain and obtain an approximation that way. The Runge-Kutta methods are a popular choice.

1.2.2 Convergence and Error Estimates

FEM is an extremely powerful technique because it can be applied to any problem of the form 1.1 if a variational form can be defined, and it can be shown that all problems have such a formulation [8]. There may be some practical difficulties along the way, but the sophistication of simultaneous equation solvers means that the most computationally intense problems can be scaled to work on high-performance clusters.
One natural question that arises is that of convergence. Convergence of the method can be proved by showing that the error of our approximated solution tends to 0 as the mesh size increases. A natural way to measure error is using the L^2-norm. This is because the Sobloev space H^1 is actually a Hilbert space, defined with the L^2-norm as its inner product. Thus, the L^2-norm is defined for all elements of V_h. The L^2 inner product of two functions f and g is defined as

$$\langle f, g\rangle_{L^2(\Omega)} = \int_\Omega f\bar{g}\,\mathrm{d}x$$

Where $\bar{g}$ is the complex conjugate of g. Then, the L^2-norm is the square root of the inner product of a function with itself, denoted

$$\|v\|_{L^2(\Omega)} = \sqrt{\int_\Omega |v^2|\,\mathrm{d}x}$$

An enlightening example would be $\mathbb{R}^n$, which is also a Hilbert space when equipped with the familiar "dot" product as its inner product. The norm can then be though of as the distance between two points if the following is calculated,

$$\|x - y\| = \sqrt{(x-y)\cdot(x-y)} = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

This is precisely the definition of distance between two points. H^1 works the same way, only with a different notion of "distance", the L^2-norm $\|u - v\|_{L^2(\Omega)}$. Then, the error of our approximation u_h from the actual solution u can be measured by $\|u - u_h\|_{L^2(\Omega)}$. In Bengzon and Larson's textbook on the finite element method[3], they find the best estimate of error to be

$$\|u - u_h\|_{L^2(\Omega)} \leq Ch^2\|D^2u\|_{L^2(\Omega)} \tag{1.4}$$

Where C is a constant, D^2u is the second order total derivative of u and h is defined to be the longest edge of any cell in the mesh, which is a way of measuring the mesh size. It is clear from 1.4 that as $h \to 0$, the error also tends to 0, which means the finite element converges as the size of the cells in the mesh decreases.

1.2.3 Overview of Tools

Finite element analysis is ubiquitous throughout science because it is so powerful. However, performing the steps outlined in Section 1.2.1 can be quite involved, and if one doesn't have a strong background in mathematics and computer programming, an implementation would be difficult to produce. Therefore, many tools have been developed to allow scientists of all fields to use FEM. These range from software libraries and packages, to bespoke programs designed for the most specific PDEs. Each have their benefits and advantages, which will be reviewed now.

PolyFEM [9] is a simple and lightweight implementation of FEM. It is written in C++, like many of libraries used today, but also has a Python binding. The power of PolyFEM is in its simplicity. It abstracts the method of solving completely, and all that is required of the user is the definition of a mesh and the parameters of the problem. Creating a mesh is straightforward, one can use an array manipulation library like Numpy for simple domains, and for more complicated domains a mesh generation package like Gmsh can be used. In addition to these options, custom-made meshes can be created using external software and stored in `.OBJ` files, which PolyFEM also accepts. An example of the parameters required can be seen in Appendix C, Listing 1. The trade-off here is of course versatility. PolyFEM only currently supports 10 PDEs, and within that customisation is limited.

Deal.II [10] is a C++ library for solving partial differential equations using the finite element method. It is one of the most modern implementations and includes many advanced methods. It is used widely in academic and commercial applications due to its versatility. While this is one of the strongest FEM libraries in the public domain, its drawback lies in the complexity of use. Their extensive documentation is well written, but extremely long due to the number of available features. This module is recommended for more advanced users. The advantage to having a FEM module written in a C-like language is that C allows programmers to have very strict control over their memory management. This allows for greater efficiency when performing calculations, leading to an overall increase in speed for the generation of meshes, and solving of systems of equations, for instance.

FreeFEM [11] is another open-source library used for the development of finite element codes. It is based in C++, and contains pre-built solvers for many popular PDEs. It interfaces with some of the most popular mesh generation software, visualisation programs, and linear algebra including Gmsh, Mumps, PETSc and ParaView. FreeFEM also comes with it's own scripting language FreeFEM++, which is used for the creation of more specific finite element solvers. Once, again the documentation is extensive, but rigorous and contains many tutorials to help users get started.

FEniCSx [12] is another open-source computing platform written in C++, with a Python binding. The mission statement explains it aim to enable users to "quickly translate scientific models into efficient finite element code". FEniCSx is written in a way that keeps code very close to its mathematical notation, meaning it is the perfect tool for users with a mathematical background. It is a package composed of four main modules. `DOLFINx` is is the main computational backend of FEniCSx. It is the backbone that connects all of the other libraries together and contains the majority of classes and functions. It contains interfaces for input/output, mesh generation, equation solvers, manipulation of finite element function spaces, and much more. `UFL` stand for Unified Form Language and is the language used to express variational forms of PDEs. Its syntax is designed to be as close to writing the actual mathematics as possible. `FFCx` is the FEniCSx Form Compiler. It takes a variational form in UFL and compiles it into a system of equations. `Basix` deals with the finite elements themselves. It can generate the basis functions of some space and their derivatives, as well as perform interpolation, projection and more.

Aside from the libraries listed above, there are many bespoke programs available as well. Specifically, there are programs made with a specific PDE in mind that they allow the user to solve. This may be for fluid dynamics, elastic deformation, geophysics, or any number of others. The benefit to these, is that it allows a user to solve problems by only inputting parameters to their equation, or specific solving options. Visualisation often comes built into these programs through a GUI. They significantly lower the barrier to entry for finite element analysis, but their main problem is that they provide little to no variability in terms of the equations themselves. However, for specific practical purposes, and for known problems, they are excellent. Another drawback is that many of these programs are proprietary, and some provide a financial barrier to their use.

1.3 Overview of Literature Regarding Mathematical Education

In order to write the learning resources effectively, a concrete understanding of the Finite Element Method is clearly needed. However an equally important part of writing a good learning resource involves having a good knowledge of education and the utilisation of an approach that puts the student experience first. We will now discuss the different approaches in education and ultimately, how they apply to mathematical education.

1.3.1 Bloom's Taxonomy

In 1956, a group of researchers from the United States published arguably one of the most influential resources in the field of education. This resource was called Bloom's Taxonomy [13], named after Benjamin Bloom, the leader of the team. They devised a hierarchical classification of learning objectives. More specifically, it was a tiered structure of skills one must obtain in order to master a subject. The hierarchy reflects how difficult each skill is to obtain, and the progression a student would take to obtain this mastery.

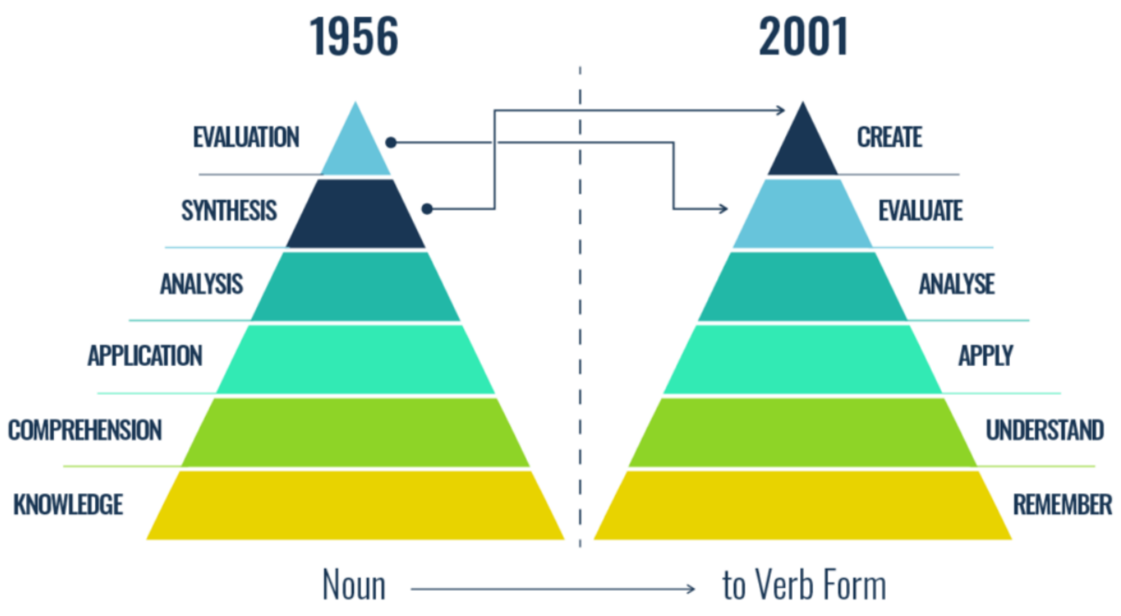


Figure 1.1: Comparison between Bloom's original Taxonomy and the revised version [14]

The original classification can be seen on the left hand side of Figure 1.1. Each layer dictates that certain tasks will be able to be completed once the layer has been mastered. These are the skills obtained.

Knowledge is the most basic and fundamental skill one must obtain. This objective is the backbone for all others, one needs to be able to remember requisite facts in order to obtain more complex understanding. Some skills gained from this level could be: definition, description, identification, and recollection.

Comprehension is the next level, it suggests a student needs more than to just blindly recall facts, they need to understand a certain logic behind them. They need to be able to describe the meaning behind statements and explain this to someone. Skills gained in this tier include: description, explanation, summarisation, and discussion.

Application is the execution of a certain procedure. This is the first part of the Taxonomy where students may not know the answer directly, instead they must use their prior knowledge to solve certain problems. Necessary skills to prove application has been obtained might include: execution, calculation, implementation and manipulation.

Analysis argues students should be able to break information into parts and draw conclusions from connections between these parts. This might be different topics in a module, or points in an argument. This is more advanced than application, since that skill was also on the recollection of steps of a procedure. Here students must generate their own conclusions, generating novel information. Skills gained here include: categorisation, comparison, deduction and criticism.

Synthesis is the next learning objective. This describes the process of creating new information. It involves organising knowledge in new ways to create a new piece of information, or a product. This skill may be required, in part, in the previous tiers, but truly creating something new is a skill all in its own. Skills that fall under this category might be: design, development, planning and invention.

Evaluation is the final learning objective in the 1956 version of Bloom's Taxonomy. In Bloom's handbook, evaluation is defined as "the making of judgments about the value of ideas, works, solutions, methods, material, etc." More plainly, it is the skill of critiquing things, something that can only be done effectively when your knowledge is at the most sophisticated level. This is why Bloom and his team put it at the peak of the taxonomy. Some of the skills one must be able to demonstrate in order to prove evaluation is satisfied, and hence a subject has been mastered are: justification, criticism, judgement and assessment.

This describes the full taxonomy when it was created. It can be clearly seen that each one flows into the next, but there is also some overlap between these levels. For instance, the skill of description could fall under knowledge or comprehension, since a student could remember a certain description which shows knowledge, but could also suggest understanding if it wasn't blindly quoted. This is why the taxonomy is not a strict rule-set for learning. Instead, it is a measurement of progress and any curriculum using it should be implemented with care, instead of blindly following the pyramid.

1.3.2 Revised Bloom's Taxonomy

Much later, in 2001, Lorin Anderson led a new team of educators to update the taxonomy, called the Revised Bloom's Taxonomy (RBT) [15]. The structure, and differences between the original taxonomy can be seen in Figure 1.1. They opted to simplify some of the language, breaking down one potential barrier to the taxonomy's use. They also opted to replace the abstract verbs with present tense versions of the same words. More importantly, it was decided that creation was a more complex skill than evaluation.
In addition to the rearrangement of learning objectives, a new dimension was added to make a so called "Taxonomy Table". This new dimension is called the knowledge dimension, while the original is called the cognitive process dimension.

Knowledge	**Cognitive Process**					
	Remember	Understand	Apply	Analyse	Evaluate	Create
Factual		░	░	▒	▒	▓
Conceptual	░	░	▒	▒	▓	▓
Procedural	░	▒	▒	▓	▓	█
Meta-Cognitive	▒	▒	▓	▓	█	█

Figure 1.2: Taxonomy Table for Bloom's Revised Taxonomy

Figure 1.2 shows the taxonomy table, with the new knowledge dimension. The shading refers to the complexity of each skill, darker being more complex. The knowledge dimension refers to the *type* of knowledge used at that level, while the cognitive process dimension refers to *how* that knowledge should be used. For instance, for one to have mastered the apply column, they must be able to apply factual, conceptual, procedural, and meta-congitive knowledge. **Factual** knowledge refers to the most basic information a student must be familiar with. It includes simple definitions, terms and symbols. **Conceptual** knowledge is the kind of knowledge one gains by understanding relationships between definitions. A lot of mathematics lies here, laws and theorems are examples of this kind of knowledge, but scientific theories, and more advanced, composite definitions also belong here. **Procedural** knowledge refers to processes. This is the knowledge needed to perform algorithms and tasks. It also includes techniques, methods and the understanding of when such processes need to be performed. Finally, the most advanced type of knowledge is **meta-cognitive** knowledge. This refers to knowledge about one's own learning. It includes the understanding of a student's own strengths and weaknesses, how they learn best, and even how to teach others effectively. It also contains knowledge about knowledge itself, for instance the structure of material in a course or material that will appear on certain tests.
Another problem Anderson and his team set out to solve the potential ambiguity within each of the definitions. They subdivided each item in the knowledge dimension into two or three "sub-types" and also divided each of the cognitive processes into categories, of which there were anywhere from two to seven. For instance, within analyse there are the categories of "differentiating", "organising", and "attributing". This would certainly help any potential educator planning to use the taxonomy table. Needless to say, it was a rigorous overhaul and

one of many overhauls Bloom's taxonomy has received since its original publication.

1.4 Learning Resource Requirements

With the aid of Bloom's taxonomy, and a few other sources it should be possible to outline the requirements of a good learning resource. These will be the guidelines will be attempted to be met, and will be evaluated against in the Results Section. The definition of learning resource used here is specifically an article, textbook, or tutorial used for self-teaching. Not included here are things like videos, or spoken word, which have their own additional criteria. See Section **??** for more information on other possibilities that could be explored.
These are 9 aspects that contribute to the effectiveness of a learning resource under this definition:

- **Good readability**
  - Readability is a measure of how easy a text is to read. A highly readable text will have shorter sentences and simpler vocabulary. If a text is readable, it means that information can be conveyed more directly allowing otherwise difficult ideas to be explained simply. One should note however, that academic subjects do have a high level of complexity and often cannot be explained in a particularly readable way. Therefore, the goal is to maximise readability, while still conveying all the intended information.
- **Effective use of visual learning methods**
  - One way to aid a student in understanding difficult ideas is through visualisation. Color, graphs, flow charts, diagrams, and more, all fall under the umbrella of visual learning approaches. They provide a new way of thinking about certain ideas, allowing readers to more easily pick up the information [16].
- **Professional aesthetics**
  - While aesthetics are not all that important and the information and explanations are the key content, they are still relevant when listing important attributes of a good learning resource. This is because information on the page should not be cluttered or distracting, the focus should be on content such as text, diagrams, animations and code snippets.
- **Good pacing**
  - A good learning resource shouldn't move too quickly through content, otherwise students may feel unprepared for new concepts. A fair amount of time should be spent on examples and explanations in order for students to fully consolidate the information. Care should also be taken however, to not slow down the pace too much. Spending too much time on an individual idea might cause the student to lose interest or start skipping through sections.
- **Proof of usefulness**

  - The ultimate aim of a learning resource is for a student to take the knowledge they gain from it and gain a practical skill. Even some of the most pure mathematical fields have practical applications. Students want to know how the information presented actually links to real problems. [16]

- **Well-tuned difficulty**
  - This section certainly links to pacing, the content should not feel difficult because the student is unprepared. However, another point to note is that any exercises provided should not be too difficult, while providing a challenge at the same time. There should be a range of questions that test the student in the subject matter, having a level of difficulty that ranges as far as the student is willing to go.

- **Sensible structure**
  - The flow of the learning resource should make sense. Not only from paragraph to paragraph but from section to section. One should motivate the other, keeping the reader engaged and causing them to continue reading.

- **Clear objectives defined**
  - At the start of the resource, clear learning objectives should be set out. This would motivate the student to continue reading, since they will not only know what they will learn, but also how they will be able to apply it. Then, at the end, students will be able to reflect on these objectives and decide for themselves whether or not they have achieved them.

1.5 Analysis of Existing FEM Learning Resources

With a more precisely defined criteria of what makes a good learning resource, it is now possible to analyse some of the most popular learning resources in finite element analysis.

1.5.1 Bengzon and Larson's Finite Element Textbook

We start with a textbook written by Mats. G. Larson and Fredrik Bengzon called The Finite Element Method: Theory, Implementation, and Applications [3]. This textbook is meant to introduce students to the finite element method by explaining the necessary mathematics, while also showing how to implement it in MATLAB.
The textbook seems to meet most, if not all criteria listed above. It has decent readability and aesthetics. Visuals have been used well throughout, the diagrams of finite element basis functions are particularly useful. The difficulty is kept to a minimum. This is due to excellent pacing that begins by introducing the problem in a 1D example, then moves up to 2D, giving readers ample time to digest the methods presented. These first five chapters give enough information for a student to be able to implement most simple PDEs. Information after this gets more advanced, and several chapters are devoted to specific engineering problems. This also gives the reader some real world context to FEM, making the reading significantly more engaging.

At the end of each chapter a selection of exercises are presented. These are very well written, allowing the reader to consolidate information gained in the previous chapter. Bloom's Taxonomy seems to be mostly satisfied, there are questions relating to simple recollection and use of definitions (recall), and to applying methods learned to specific problems (comprehension and application). In questions that require the reader to construct a proof, analysis is met because, one must bring together several concepts and ideas to construct an argument. Synthesis could also be considered to be part of proof-writing, although the definition might be being stretched here. Evaluation however, is not so well explored. This could be one criticism against the book. Bengzon and Larson could have asked students which approach might be better for certain problems for instance, which would force the reader to weigh up the advantages and disadvantages of certain methods.

1.5.2 J. S. Dokken's FEniCSx Tutorial

The next FEM learning resource is a tutorial specifically written as a guide for FEniCSx, the previously mentioned FEM computing platform. It is written and maintained by Jørgen S. Dokken, a researcher in the field and member of the FEniCSx Steering Council. The tutorial was adapted from Langtangen and Logg's book *Solving PDEs in Python: The FEniCS Tutorial I* [?], which was written for the previous version of FEniCSx.
Rather than as a guide to the finite element method, this tutorial serves more as a guide on how to use FEniCSx. Having said that, mathematical explanations are provided, they are just simply not as detailed as Bengzon-Larson for instance. The tutorial is written using in a Notebook format, which is spoken about in Section **??**, but means that code snippets feature heavily. Dokken uses a service called Binder to host the notebooks remotely, which allows users to run the code in their own browsers without downloading any dependencies. This removes a large barrier to the use of FEniCSx, and lets users tinker and get familiar with the library without having to download the libraries on their local machine. If one does not wish to use Binder, they are free to view the static version as well.
The notebook is very readable, jargon is kept to a minimum, although there is still quite a lot due to the subject matter. Many terms are hyperlinked to external sites where readers can find more detailed information. Visuals are used vary sparingly, which is one downside of the tutorial. It could certainly be improved with a few diagrams or illustrations. The pacing is excellent, and each chapter uses information from the previous to expand the reader's knowledge in an engaging way. Sometimes the pace can be slightly too fast, and important concepts don't have enough time spent on them, but in these cases external resources are often linked to, and Dokken recommends that readers unfamiliar with FEM read a textbook on the subject.
The usefulness is motivated fairly well, there is a section titled "Gallery of Finite Element Solvers" which provides plenty of examples. Most of the problems presented however seem to be idealised, but still provide good insight into the potential applications of FEniCSx. The sections flow well between each other, and difficulty is well managed. Learning objectives also feature at the start of every chapter, which is great to see.

Chapter 2

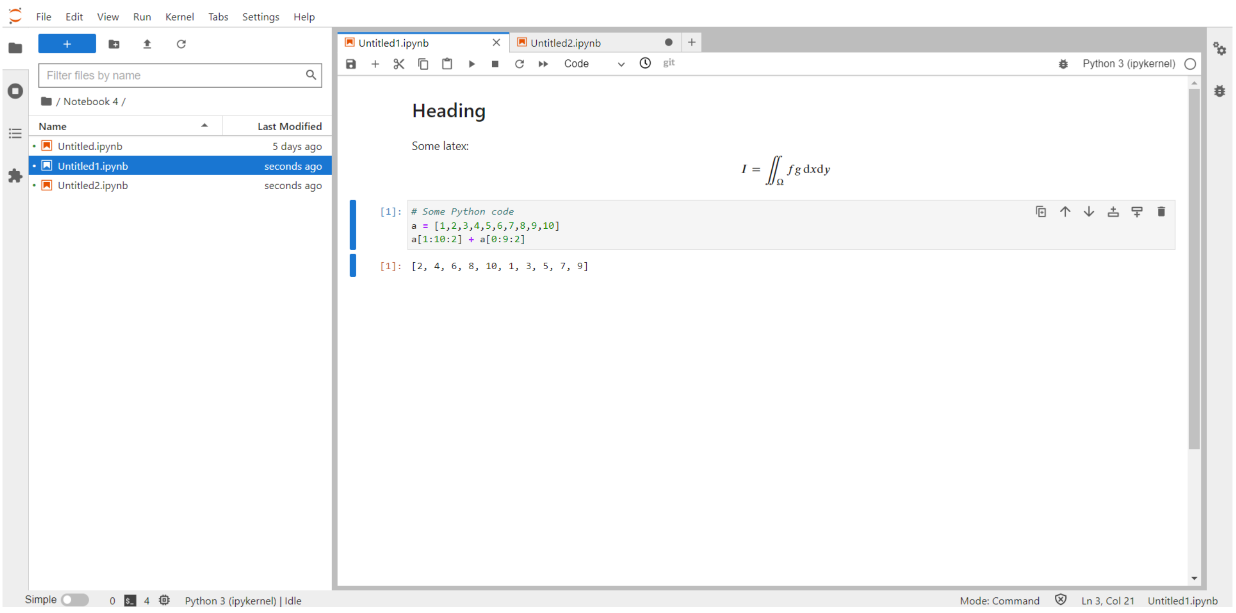
Methods

2.1 Software Design

With the necessary background research performed, the next step was to design the architecture of the online learning resource.

2.1.1 JupyterLab

JupyterLab is an interactive computing platform developed by Project Jupyter [?]. It allows users to run code inside their browsers, as well as displaying Markdown and LaTeX. Markdown is a simple language for formatting text. It includes things like headings, bullet points, tables, images etc. It is used widely in documentation, readme files and other educational coding resources. It is incredibly lightweight and easy to write, and it's philosophy is about putting readability first [17]. LaTeX is another so called "mark-up" language widely used in academic writing. It can be used to construct professional looking documents, and gives the user a massive amount of control in their construction. It also has an extensive set of mathematical symbols, and environments, making it one of the best tools for writing mathematics on a computer. JupyterLab uses a JavaScript library called MathJax [18] to implement the mathematical functionality of LaTeX.
The interface of JupyterLab can be seen in Figure 2.1, the main section on the right is where code and markdown is displayed, while the left shows other options. In this figure, it is showing the file tree, although this can also show kernels currently in use, a structured view of the notebook currently open and any extensions.



Heading

Some latex:

$$I = \iint_\Omega f g \,\mathrm{d}x\mathrm{d}y$$

```
[1]: # Some Python code
     a = [1,2,3,4,5,6,7,8,9,10]
     a[1:10:2] + a[0:9:2]

[1]: [2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
```

Figure 2.1: The JupyterLab Interface

Jupyter is an excellent way of running code for instructional or demonstrative purposes. The

LaTeX allows for precise mathematical descriptions, while Markdown keeps everything readable and clean. It is also very quick to write in. Jupyter supports most modern languages by allowing the user to specify a kernel, although the main languages that are supported are Julia, Python and R (Which are the origins of the name Jupyter). In this context, a kernel is the computational engine used to run code within the notebook, of which there are many official versions and community-maintained versions.

2.1.2 Python

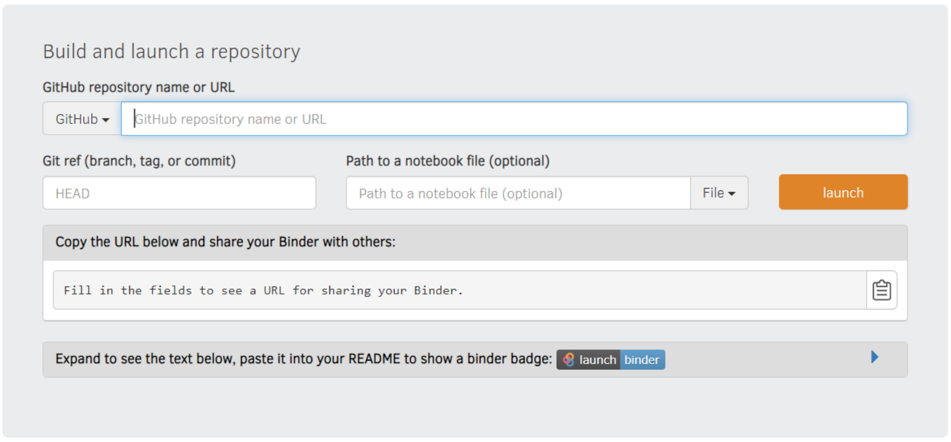
The Finite Element library used is called FEniCSx, which was introduced in Section 1.2.3. The justification behind using this particular platform, is that it provides a good compromise between versatility and complexity. FEniCSx is very versatile because has a lot of features, meaning PDEs arising in any scientific discipline can be solved. It is also easier to pick up than something like FreeFEM++, since it is based solely on Python and there are some good already tutorials already created. In FEniCSx, the most basic PDE instances can be defined, solved and visualised in around fifty lines.
One of the downsides of using FEniCSx is that it has a lot of dependencies. The guide to installing all the source packages can be seen here. At least eleven separate libraries are required for the C++ core, after which one can install the Python interface. Needless to say, this is a time consuming and complicated process. It can be streamlined greatly by instead using Docker.

2.1.3 Docker

Docker is a virtualisation software that allows users to run virtual machines at an operating system level. Each virtual machine is called a container, which is designated its own an area of memory. When creating a docker container, one must specify its contents. This is done by creating an image, a kind of template that containers copy from. Images contain a set of instructions that create directories, download and install programs, and generally set up a container for use. Docker containers are essentially separate, independent environments in which one can work. The point of having these separate environments is to allow for the development of independent software features. As well as it's use for software development, it can also be used to run complicated dependencies. An organisation, such as the Fenics Group, might host a Docker image from which any user can pull. Once the user has the Docker image locally, they are able to create their own Docker containers and work with the organisation's software. The FEniCSx group has such an image! It can be found on the DOLFINx Github page.
However, while these docker images are excellent and contain the whole of FEniCSx and more, they don't contain everything necessary for our learning resources. For instance the plotting library Matplotlib is not included in the main Docker image. Also certain timing and meshing libraries are not built in. It is possible to create a Docker image using a DockerFile. In said file, one can pull from an existing image, which is exactly what would be needed in our case. However, the simpler option here was to use a pre-made docker image hosted by Dokken for use with his FEniCSx Tutorial [19].

2.1.4 Method of Delivery

Having created the chosen development environment, a docker container inheriting from Dokken's image, the next logical thought was how to allow users to access the notebooks. One of the main features of a notebook that makes it appealing is the interactive nature of the blocks. User's can run code examples to see dynamically changing outputs. They can alter the author's code in any way they like, or use the dependencies to solve their own problems. As such, the interactive aspect needed to be kept, instead of converting the notebook files to static PDFs. The first method attempted, was to use a service called Binder [20]. Binder is a service that hosts Jupyter notebooks in an executable environment. This means anyone, anywhere can access the notebooks without setting up any dependencies. They only have to follow a link. Sounds like the perfect solution!


Build and launch a repository

GitHub repository name or URL

GitHub: GitHub repository name or URL

Git ref (branch, tag, or commit): HEAD

Path to a notebook file (optional): Path to a notebook file (optional)

File

launch

Copy the URL below and share your Binder with others:

Fill in the fields to see a URL for sharing your Binder.

Expand to see the text below, paste it into your README to show a binder badge: launch binder

Figure 2.2: Hosting a Jupyter Notebook with Binder

All Binder needs to host a notebook is a link to the repository containing said notebooks, plus a few launch options. It's incredibly simple and would work excellently if it wasn't for one problem. The notebook's dependencies can normally be specified by adding a `requirements.txt` file listing the Python/Julia/R libraries needed and their versions. However, since we have many libraries that are actually built from C++, and require more care to install, this approach will not work. Instead a DockerFile must be provided. This makes sense, because Binder actually uses a Docker container themselves to host the notebooks being run. However, despite the feature being implemented by Binder, they actually discourage users from using it. They consider it an "advanced use case" and, crucially, "cannot guarantee that the DockerFile will work" [21].

Unfortunately, this was indeed the case upon integrating Dokken's DockerFile into the repository used to host the notebooks. Commits de7cec4 through to 53e565f were all attempts to get Binder working. It was all based off of Dokken's file, which already worked with Binder in his repository, but when the exact same files were in mine, it didn't work. This was a blow, but in the interest of time, the decision was made to not go with Binder and instead get users to run the notebooks themselves.

Getting users to run the notebooks themselves required them to install Docker on their own machines and create a container themselves. The installation of Docker can be fairly involved to someone who doesn't know the process, since it requires a separate virtualisation software at an OS-level. Linux based operating systems come with native virtualisation, but Windows does not. Therefore, one of the best ways to run docker on Windows is to install Linux on Windows and then perform the containerisation inside of Linux (inside of Windows). Figure 2.3 illustrates the hierarchy.

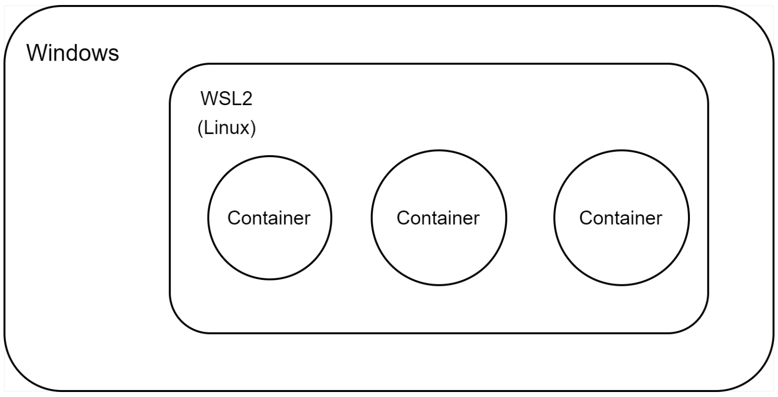


Figure 2.3: Virtualisation Hierarchy

The process of installing WSL2 (Windows Subsystem for Linux) is not too complicated, although a guide is certainly required in order to explain the process, and also how to create the container itself. Therefore, since this seemed to be the only option, a guide was created and uploaded to the repository for users convenience. The resulting PDF can be found here and describes multiple options depending on the user's system, with the goal being to allow as many students to access the material as possible.

2.2 Content Design

2.2.1 User Experience

A flow diagram of the user experience in the resource's current form is shown in Figure 2.4.

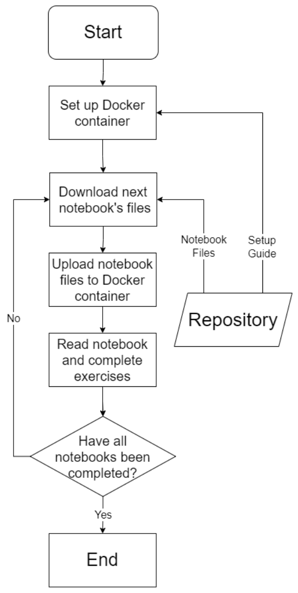
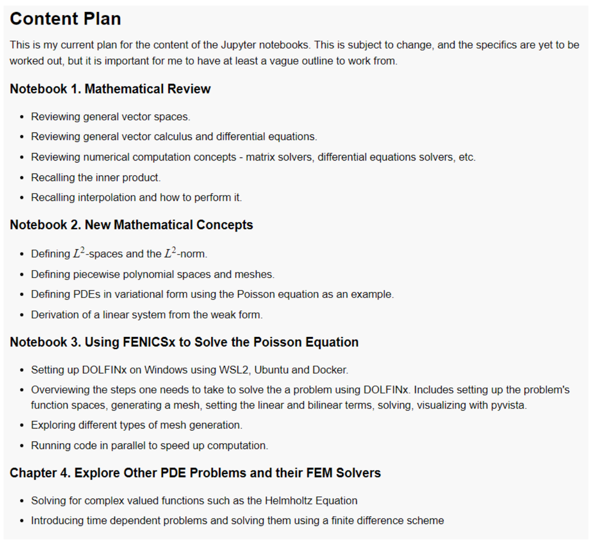

Start
Set up Docker container
Download next notebook's files
Upload notebook files to Docker container
Read notebook and complete exercises
Have all notebooks been completed?
No
Yes
End
Notebook Files
Setup Guide
Repository

Figure 2.4: Flow Diagram of User Experience

The user will begin by accessing the repository containing all the notebook files, called `FEMLearningResources`. From here, they must download and complete the setup guide to install Docker and create their Docker container. Once this is done they will have everything they need to proceed with running and viewing the notebooks in an interactive setting. Next, the user downloads the files for the notebook they are want to read. This needs to be unzipped and then should be uploaded to the Docker container. Jupyter has functionality for uploading documents, so this can be used. Finally, the user can read the notebook, run code blocks and complete exercises.
While this experience is a little more involved than the Binder approach, it does teach students how to use Docker, a very widely used tool in software development. However, it means that the barrier to entry is higher than it needs to be which is unfortunate.

2.2.2 Content Plan

With the learning structure defined, we will now discuss the design of the content plan. The initial goal was to teach students to understand and use the Finite Element Method. The belief during the design phase was that "understanding" took precedence over "using". This is because if one understands the theory behind a method, implementation is broadly the same over different specific libraries. Furthermore, FEniCSx is a library very closely related to the mathematical description of FEM, so a focus on the mathematics would lead more naturally into this. Another balance that was considered was between simplicity and complexity. Making the content too simple would not meet the goal of the project and would leave students feeling dissatisfied. Making the content too complex by increasing the scope would put pressure on the time constraints of the project, and may lead to students feeling rushed or unsure about certain topics due to the less considered development. This is how to original content plan was developed. It can be seen in Figure 2.5a.



Content Plan

This is my current plan for the content of the Jupyter notebooks. This is subject to change, and the specifics are yet to be worked out, but it is important for me to have at least a vague outline to work from.

Notebook 1. Mathematical Review

- Reviewing general vector spaces.
- Reviewing general vector calculus and differential equations.
- Reviewing numerical computation concepts - matrix solvers, differential equations solvers, etc.
- Recalling the inner product.
- Recalling interpolation and how to perform it.

Notebook 2. New Mathematical Concepts

- Defining L^2-spaces and the L^2-norm.
- Defining piecewise polynomial spaces and meshes.
- Defining PDEs in variational form using the Poisson equation as an example.
- Derivation of a linear system from the weak form.

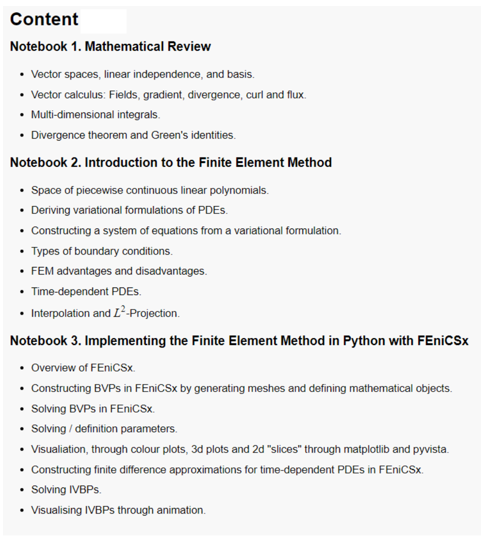
Notebook 3. Using FENICSx to Solve the Poisson Equation

- Setting up DOLFINx on Windows using WSL2, Ubuntu and Docker.
- Overviewing the steps one needs to take to solve the a problem using DOLFINx. Includes setting up the problem's function spaces, generating a mesh, setting the linear and bilinear terms, solving, visualizing with pyvista.
- Exploring different types of mesh generation.
- Running code in parallel to speed up computation.

Chapter 4. Explore Other PDE Problems and their FEM Solvers

- Solving for complex valued functions such as the Helmholtz Equation
- Introducing time dependent problems and solving them using a finite difference scheme

(a) Original Content Plan



Content

Notebook 1. Mathematical Review

- Vector spaces, linear independence, and basis.
- Vector calculus: Fields, gradient, divergence, curl and flux.
- Multi-dimensional integrals.
- Divergence theorem and Green's identities.

Notebook 2. Introduction to the Finite Element Method

- Space of piecewise continuous linear polynomials.
- Deriving variational formulations of PDEs.
- Constructing a system of equations from a variational formulation.
- Types of boundary conditions.
- FEM advantages and disadvantages.
- Time-dependent PDEs.
- Interpolation and L^2-Projection.

Notebook 3. Implementing the Finite Element Method in Python with FEniCSx

- Overview of FEniCSx.
- Constructing BVPs in FEniCSx by generating meshes and defining mathematical objects.
- Solving BVPs in FEniCSx.
- Solving / definition parameters.
- Visualiation, through colour plots, 3d plots and 2d "slices" through matplotlib and pyvista.
- Constructing finite difference approximations for time-dependent PDEs in FEniCSx.
- Solving IVBPs.
- Visualising IVBPs through animation.

(b) Updated Content Plan

Figure 2.5: Change in content between start and end of the project

The content originally spanned 4 Notebooks, each of increasing complexity and interest to the reader. The notebooks were to begin with a general mathematical review, preparing the reader

for the new mathematics that was to follow. The reader was assumed to have a mathematical knowledge of a second or third year mathematics undergraduate student and basic knowledge of coding. This influenced the pace at which new material could be introduced as well as the topics that need to be included in the review. Topics like vector calculus, linear algebra and differential equations should be familiar to them, so could be covered in a less detailed way than the FEM content. It felt important to ease the readers in with a refresher, because rushing into new advanced mathematics could lead to confusion. They may have studied the concepts a while ago, so a reminder was important. As such, an entire notebook was devoted to revising these subjects.
Next was the first notebook of new content. The content can be seen in the figure, but it included all of the basic mathematical definitions and procedures for understanding the Finite Element Method. The third notebook was intended to give the readers a chance to get to grips with the implementation of the method. Originally, this is the point the setup guide was needed. The first two notebooks did not have interactive elements and were planned to be static resources. It was decided at a later point that even non-interactive parts of the resource would be created using the notebook format.
Finally, the fourth notebook was planned to give more complicated examples, such as the Helmholtz Equation, a PDE defined for complex-valued variables. Also the modelling of time-dependent systems was to be explored. This was left partially empty due to the idea that additional sections would be added later.
When development began, the content plan had to change. This was because a greater understanding of FEM had been obtained and certain aspects were deemed unnecessary to the goals of the project. Furthermore, a more refined idea of scope was developed, leading to the eventual cut of certain sections. This led to the refinement of four notebooks, to three. The revised syllabus can be seen in Figure 2.5b. A brief explanation of changes is listed below.

- Differential equations were removed from the review since enough knowledge could be gained from the vector calculus definitions to make this irrelevant.
- A precise description of inner products was removed since it wasn't all that necessary for understanding the function spaces involved.
- Emphasis was taken away from the L^2 spaces, they would still be mentioned, but it was not all that important to devote much time to them.
- Boundary conditions and FEM advantages and disadvantages were added.
- Some content was rearranged to increase flow and motivate future sections, specifically interpolation and time-dependent PDEs.
- Certain points were made much more specific.
- Descriptions of the parallel computation, different types of mesh generation and complex-valued PDEs were all dropped due to make the scope of the project more realistic.

2.2.3 Adapting the content

In order to maintain accuracy and correctness, the specific content of the notebooks had to be derived from well-respected and accurate sources. Notebook 1 spans the broadest spectrum of topics, and as such, was derived from the most sources. Knowledge gained from studying mathematics at university came in useful, but s set of invaluable resources in refreshing my memory was the lecture notes provided by module leaders at the University of Leeds. These modules included the following.

- MATH2022 Groups and Vector Spaces
- MATH1005 Core Mathematics (Introductory calculus, linear algebra and differential equations)
- MATH2365 Vector Calculus
- MATH2375 Linear Differential Equations and Transforms

They were useful not only for the correctness and accuracy of the mathematical statements, but also for their use in explaining concepts in a natural way. While these made up most of the sources for Notebook 1, Wolfram MathWorld [22] also played a part in filling in some of the gaps where alternative or additional explanations were needed.
Notebook 2 was based primarily off Bengzon and Larson's textbook [3]. Inspiration was taken from the structure of the book. The authors decided the first two chapters should explain piecewise polynomial spaces and the finite element method in 1 dimension only. This means the domain being solved over was an interval on the number line. Since there is only one variable to differentiate with respect to, this simple case is an ODE, not a PDE. The simplicity enabled the reader to build upon the base knowledge presented and progression to higher dimensions followed very naturally.
Most of Notebook 3's material was taken from Dokken's tutorial [23]. His tutorial is one of the only learning resources on FEniCSx available and is by far the most accessible and well-written. Due to this fact and also due to the complexity of the library, a large portion of the code presented in Notebook 3 was written by Dokken. However, the explanations he gives in the tutorial can sometimes be a little shallow, and care has been taken to expand and enlighten. A particularly interesting tangent was on bounding-box trees, a type of data structure that can be used to sort objects in space by the size of their bounding-box. An excellent source for this was this blog post by James from Azure From The Trenches [24]

2.2.4 Notebook Structure

The structure of the notebooks was laid out much like a book, with distinct sections and subsections of content separated with exercises. Each notebook was split into two or three sections each tackling a distinct area of study. Notebook 1 contains **Vector Spaces** and **Vector Calculus**. Notebook 2 contains **FEM in 1D**, **FEM in 2D**, and **Time Dependent Problems**. Notebook 3 contains **Introduction to FEniCSx** and **A Time Dependent Example**. Within these sections, there are a varying number of subsections that break the content up into more manageable topics. These can be seen in Figure **??**.

Development of the exercises was another important part of the design. Exercises serve to test understanding and cement newly ingested information. They can also be great at challenging possibly incorrect assumptions students may have picked up from the content. After each subsection in a notebook, there were a collection of "check" exercises. These ranged from simple calculations, to proving statements in the text, to explanations and sometimes more complicated derivations. For the computing side, exercises were harder to prescribe for each subsection, since an entire section spanned a single example. However, at the end of each section, a chance to apply the methods learned was given through way of practise question. There were even exercises that linked between each other in building a full PDE approximation, which was intended to leave a sense of learner satisfaction in creating their own approximation the whole way through.

2.3 Development Methodology

Throughout the development of the project, version control was used to store

Chapter 3

Results

3.1 User Feedback

In order to evaluate the success of the solution produced, a method of collecting user feedback had to be developed. This was done by collecting a group of eligible testers who not only had relevant background to understand the material, but were also willing to devote time to the study and review the material. Then, a consistent and precise way of collecting the information had to be created so that information regarding how successful

3.2 Notebooks

3.2.1 Notebook 1

3.2.2 Notebook 2

3.2.3 Notebook 3

Chapter 4

Discussion

4.0.1 Analysis

** Compare their responses with some resource I can find online. Maybe an empirical study has done something similar. As always, provide plenty of references. As a result of the comparison, evaluate what was successful and what was unsuccessful about the project. Could compare against more advanced online educational services (codecademy, brilliant, khan etc). **

4.1 Conclusions

** Did I meet the project goals set out at the start? Do I fully know if it was a success or a failure and why? What would I do differently if I were to do it again? **

4.2 Ideas for future work

** Possibility for using more sophisticated means of teaching. Deeper exploration of Finite Element Analysis, maybe room for a part 2 to the course? **

References

[1] Michael Renardy and Robert C. Rogers. *An Introduction to Partial Differential Equations*. Number 13 in Texts in Applied Mathematics. Springer=Verlag, New York, 2 edition, 2004.

[2] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Number 15 in Texts in Applied Mathematics. Springer Science + Business Media, New York, 2008.

[3] Mats G. Larson and Fredrik Bengzon. *The Finite Element Method: Theory, Implementation, and Applications*. Number 10 in Texts in Computational Science and Engineering. Springer-Verlag, New York, 2013.

[4] W. Ritz. Über eine neue Methode zur Lösung gewisser Variationsprobleme der mathematischen Physik. *Journal für Reine und Angewandte Mathematik*, 135, 1908.

[5] J. Singer. On Equivalence of the Galerkin and Reyleigh-Ritz Methods. *Journal of the Royal Aeronautical Society*, 66(621):592, 1962.

[6] M.J. Turner, R. W. Clough, H.C. Martin, and L.J. Topp. Stiffness and Deflection Analysis of Complex Structures. *Journal of the Aeronautical Sciences*, 23(9), 1956.

[7] I. Babuska and A.K. Aziz. Lectures on mathematical foundations of the finite element method. Technical report, United States, 1972.

[8] E. Toni. Variational formulation for every nonlinear problem. *International Journal of Engineering Science*, 22(11/12):1343–1371, 1984.

[9] Geometric Computing Lab. PolyFEM. https://polyfem.github.io, NYU Courant Institute of Mathematical Sciences and the University of Victoria, Canada, 2023.

[10] Daniel Arndt, Wolfgang Bangerth, Marco Feder, Marc Fehling, Rene Gassmöller, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Simon Sticko, Bruno Turcksin, and David Wells. The `deal.II` Library, Version 9.4. *Journal of Numerical Mathematics*, 30(3):231–246, 2022.

[11] F. Hecht. New development in FreeFem++. *Journal of Numerical Mathematics*, 20(3-4):251–265, 2012.

[12] FEniCS Project. FEniCSx. https://fenicsproject.org/, 2023.

[13] Benjamin S. Bloom (Ed.), Max D. Englehart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. *Handbook 1: Cognitive Domain*. Taxonomy of Educational Objectives - The Classification of Educational Goals. David McKay Company, Inc., 1956.

[14] Growth Engineering. Bloom's Taxonomy : Master Your Learning Objectives. https://www.growthengineering.co.uk/what-can-blooms-taxonomy-tell-us-about-online-learning/, 2022.

[15] Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Rintrich, James Raths, and Merlin C. Wittrock. *A Taxonomy for Learning, Teaching, and Assessing - Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman, Inc., 2001.

[16] Layal Hakim. Top tips to improve the teaching of mathematics in universities. https://www.timeshighereducation.com/campus/top-tips-improve-teaching-mathematics-universities, Times Higher Education, University of Exeter, 2022.

[17] John Gruber. Daring Fireball | Markdown: Syntax | Philosophy. https://daringfireball.net/projects/markdown/syntax#philosophy, 2013.

[18] Davide Cervone and Volker Sorge. MathJax. https://www.mathjax.org/, 2013.

[19] J. S. Dokken. DOLFINx Tutorial Docker Image. https://github.com/jorgensd/dolfinx-tutorial#docker-images, GitHub, 2023.

[20] Jupyter et al. Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale. https://conference.scipy.org/proceedings/scipy2018/project_jupyter, 2018.

[21] Jupyter et al. Binder User Guide | Use a Dockerfile for your Binder repository. https://mybinder.readthedocs.io/en/latest/tutorials/dockerfile.html#preparing-your-dockerfile, 2022.

[22] Wolfram. Wolfram MathWorld. https://mathworld.wolfram.com/.

[23] Jørgen S. Dokken. The FEniCSx tutorial, 2022.

[24] James. Introductory Guide to AABB Tree Collision Detection. https://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-collision-detection/, Azure From The Trenches.

[25] Geometric Computing Lab. PolyFEM JSON Files Structure. https://polyfem.github.io/json, NYU Courant Institute of Mathematical Sciences and the University of Victoria, Canada, 2023.

Appendix A

Self-appraisal

<This appendix should contain everything covered by the 'self-appraisal' criterion in the mark scheme. Although there is no length limit for this section, 2—4 pages will normally be sufficient. The format of this section is not prescribed, but you may like to organise your discussion into the following sections and subsections.>

A.1 Critical self-evaluation

A.2 Personal reflection and lessons learned

A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

A.3.1 Legal issues

A.3.2 Social issues

A.3.3 Ethical issues

A.3.4 Professional issues

Appendix B

External Material

Appendix C

Reference Material

This Appendix is to reference material that was not used in my implementation, but I came across while researching potential solutions.

```
{
    "common": "", // path to another JSON file containing default
                  // arguments on which to patch these arguments

    "geometry": [{
        "mesh": "" // mesh path (absolute or relative to JSON file)
    }],

    "time": {                           // time-dependent problem
        "tend": 1,                      // end time
        "dt": 0.1,                      // time step size
        "time_steps": 10,               // (alternativly) number of time steps
        "integrator": "ImplicitEuler" // time integration method
    },

    "contact": {
        "enabled": true // enable contact handling
    },

    "solver": {
        "linear": {
            "solver": "Eigen::PardisoLDLT"
        },
        "nonlinear": {
            "line_search": {
                "method": "backtracking"
            },
            "solver": "newton"
        }
    },

    // Material parameter
    "materials": {
        "type": "NeoHookean", // material model
        "E": 1.5, // Young's modulus
        "nu": 0.3, // Poisson ratio
        "rho": 1 // density
    },

    "output": {
        "json": "sim.json",             // output statistics
        "paraview": {                   // output geometry as paraview VTU files
            "file_name": "sim.pvd",
            "options": {
                "material": true,       // save material properties
                "body_ids": true        // save body ids
            },
            "vismesh_rel_area": 1e-05 // relative area for upsampling the solution
        }
    }
}
```

Listing 1: PolyFEM JSON Structure Example [25]