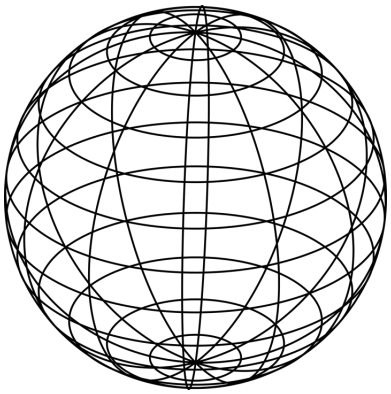
Approximating the element matrices in an unfitted finite element method using neural networks

By

AMANDA BERTGREN



Master of Sciences Thesis in Engineering Physics, 30 ECTS credits
UMEÅ UNIVERSITY

13th June 2022
Umeå, Sweden

Approximating the element matrices in an unfitted finite element method using neural networks

Amanda Bertgren (amandabertgren@outlook.com)

Supervisor: Karl Larsson (karl.larsson@umu.se)
Examiner: Mats G. Larson (mats.larson@umu.se)

Thesis project, 30 ECTS credits
Master of Sciences in Engineering Physics

Department of Physics
Umeå University
Universitetstorget 4, 901 87 Umeå

”The brave things in the old tales and songs, Mr. Frodo, adventures, as I used to call them. I used to think that they were things the wonderful folk of the stories went out and looked for, because they wanted them, because they were exciting and life was a bit dull, a kind of a sport, as you might say. But that’s not the way of it with the tales that really mattered, or the ones that stay in the mind. Folk seem to have been just landed in them, usually their paths were laid that way, as you put it. But I expect they had lots of chances, like us, of turning back, only they didn’t. And if they had, we shouldn’t know, because they’d have been forgotten.”

- J.R.R. Tolkien, 1954. *Lord of the Rings: The Two Towers*.

ABSTRACT

This study investigates the possibility of combining an unfitted finite element method, CutFEM, with neural networks, in an attempt to reduce the computational time on evolving domains. Finite element methods are used to solve partial differential equations by fitting a spatial and temporal discretisation to the momentous domain. CutFEM was proposed to overcome a repeated discretisation of evolving domains by introducing a static background mesh allowed to cut the domain boundary. The drawback is an increased complexity in quadrature of the cut contributions to the element matrices, which quickly becomes time consuming for higher order methods.

As machine learning methods have been successful in a variety of areas recently, this study investigates the possibility of replacing the element matrix quadrature with neural network regression. A classification network is also proposed for a quadrature-free method of identification of the cut elements. The study has been performed by implementing the networks in alternative CutFEM algorithms. Different methods of implementation, pre- and post-processing of the data, as well as different optimisation strategies of the training phase have been investigated in comparison to CutFEM benchmarks.

The time consumption showed a significant decrease for the modified CutFEM in comparison to the conventional method. The classification was shown to be successful for two elementary domains, yet some difficulties occurred for a non-trivial level set representation. The approximation of cut elements with sufficient domain intersection showed decent results, although elements with minor intersection with the domain contributed with some difficulties and the error tends to propagate towards nearby elements. This was deduced to stem from the nodal contributions furthest from the domain intersection. Therefore, this study proposes a few additions to the current method in the event that the study is extended. These suggestions are based around the introduction or modification of error weights in the neural network training.

Keywords

Finite element method, CutFEM, neural networks.

ACKNOWLEDGEMENTS AND DEDICATIONS

I want to thank my supervisor Karl - both for sharing his knowledge within the field of finite element methods as well as providing me with guidance and general insights during this period.

I wish to dedicate this work to my family and close ones. The following is in Swedish so that it can be perceived by those mentioned.

Till Sandra och Zita. Tack för att ni inspirerar mig.
Till mamma och Mathias. Tack för allt ert stöd.
Till mummo, morfar och Lalla. Tack för att ni alltid tror att jag kan.

TABLE OF CONTENTS

LIST OF TABLES

TABLE **Page**

LIST OF FIGURES

FIGURE **Page**

CHAPTER 1

INTRODUCTION

Finite element methods (FEMs) are commonly used to solve systems of partial differential equations (PDEs). These systems typically arise in a wide range of engineering applications, such as heat conduction or diffusion processes. Ideally, an analytical solution would be obtained, however, this can rarely be achieved due to high complexity of the system and finite computational performance. Instead, an approximate numerical solution can be derived using FEM.

The FEM computations stem from a discretisation of the domain in the spatial and temporal dimension. In the spatial dimension, this is achieved by sub-sectioning the domain into a set of elements based upon a mesh with nodal points fitted to the geometry of the domain. The elements are equipped with basis functions of chosen order to include different properties and achieve desired precision of the solution. Each of these elements can then be evaluated independently of others, and the element-wise contribution is assembled into a linear system containing the discretised properties of the entire domain expressed in terms of element matrices. Solving the linear system is a much more manageable task than solving the initial system of PDEs, and the approximation error is strictly dependent on the properties of the chosen elements.

The momentous evaluation is thus without any issue in the general case, however, some challenges are introduced if the boundary of the domain is evolving over time. More specifically, the process of discretisation must be performed for each time iteration since the mesh is fitted to the momentous domain. This is highly time consuming, and one is left with a trade-off between computational speed and precision.

An unfitted element method can be used to avoid the repeated process of discretisation. It is based upon one initial discretisation which is completely independent of the domain of the system. The elements are thus allowed to cut through the boundary, resulting in a number of cut elements with partial intersection of the domain. Its implementation and possibilities has been further examined in many applications. One of them is XFEM which extends the domain

space to allow for a solution including the cut elements, and another is CutFEM which can be considered a stabilised version of XFEM [2].

1.1 Background

This work revolves around CutFEM which has been much researched by, for example, [3, 4]. See [5] for a collection of research on unfitted FEM as well as further references to related works. CutFEM differs from conventional FEM since the solution is discretised on a background grid to the domain in the form of a static mesh, allowing the boundary of the domain to intersect the elements. The corresponding element matrix contributions are computed using quadrature over the intersection between the element and the domain. This, however, introduces a challenge to the well-posedness of the element matrices since some intersections may not be large enough to produce sufficient matrix contributions. This is often approached with a ghost penalty term, as described in [6], which extends desired properties from the domain to the elements intersecting it. Another challenge concerns the imposition of boundary values, a task which becomes non-trivial when the boundary is not aligned to the elemental nodes. This can however be overcome by a weak enforcement as first introduced for Dirichlet boundaries in [7], which is outlined for unfitted finite element methods in [8].

CutFEM thus removes the need for repeated discretisation and the primary associated challenges can be overcome as described above. However, it introduces the requirement of repeated quadrature for non-trivial contributions stemming from the intersecting elements. This quickly becomes time consuming for higher orders of the element representation. There have been many attempts to overcome this limitation, to different results. A conventional sub-discretisation with Gaussian quadrature is used in [3] which proves successful in terms of precision, yet may quickly increase the computational time for repeated computations. In [9] a reduction in the integral dimension is investigated and in [10] a variety of methods, such as Monte Carlo representation of the integral, is evaluated. The issue is however reoccurring and one is again left with the trade-off between computational speed and precision.

Artificial intelligence, especially machine learning and neural networks, has become increasingly popular since the necessity for new methods processing data of high dimensionality has grown. The ingredients to constructing a successful network do however require a specific fine-tuning of the architecture, data set and input. If a network can be sufficiently optimised, it might prove suitable to replace the step of quadrature in the CutFEM method using numerical regression. The task would essentially become a problem of integration for the network, which has been investigated in, for example, [11–13] in its conventional form. Since the results are generally successful, especially for smooth functions, it may be possible to combine the neural network integration method with CutFEM to attain a sufficient finite element approximation for problems where speed is especially important, such as real time implementations.

1.2 Outline of thesis

This thesis is structured such that it first presents the CutFEM method in terms which are relevant for this work. A model problem is introduced, which will be used as a benchmark in the comparison between the classical method and the neural network based method. It will also constitute an example in the derivation of the CutFEM method.

This will be followed by some notes on the performance of neural networks and how it can be fine-tuned in this work. This concerns some specific notes on architectural starting points as well as any methods applied on the corresponding data set and input to optimise the performance.

When the general background have been presented, the specific choices of method and implementation will be discussed. Whilst most motivations will have been outlined at this point, this section will further specify what has been performed so that any suggested updates can easily be performed with this work as a reference. A few of the running tests are also mentioned, although detailed, numerical results of these are generally excluded based on its contextual relevance.

The results of the methods of choice are then presented, along with some comments on the relation between the displayed results and the overall study.

Lastly, the results will be discussed and elaborated upon. This will allow for inspection of any reoccurring errors in the results, providing an analysis of their origin and any suggested methods of preventing them in a possible extension of this study.

CHAPTER 2

FINITE ELEMENT METHODS AND HOW THEY MAKE THE CUT

As has been indicated, finite element methods are used within a variety of engineering applications, commonly by its implementation in computed aided engineering softwares. Their purpose is to solve systems governed by phenomena which can be modelled by partial differential equations - such as wave propagation, deformation or heat conduction. The initial problem is simplified by performing an element-wise evaluation of a discretised domain. The element-wise contributions are then assembled into a linear system, which can be solved numerically to a precision related to the choice of element.

This chapter is based around the CutFEM method as illustrated for a model problem which is used in this thesis. By introducing the problem, we begin our path onto obtaining an unfitted finite element formulation with relevant theoretical results. Using some introductory definitions, we define the relevant spaces and can perform an abstract variational analysis of the problem. Moving further, the unfitted mesh and domain representation are outlined, as well as a reference mapping, which allows us to formulate a cut finite element formulation given the performed discretisation. Some mathematical finesses are demonstrated, which will prove useful later in the implementation. Lastly, stabilisation of ill-conditioned elements is discussed.

2.1 Introduction of a model problem

The Poisson problem is the most classical benchmark problem used in finite element formulations. One variation of the Poisson equation is the reaction-diffusion equation which has an extra zeroth order term. For an arbitrary domain $\Omega \in \mathbb{R}^2$ with boundary Γ, the solution $u = u(x, y)$ is sought to the problem defined by

$$-\Delta u + cu = f \quad \text{in } \Omega \tag{2.1}$$

$$\mathbf{n} \cdot \nabla u = g_N \quad \text{on } \Gamma \tag{2.2}$$

where $f = f(x,y)$ denotes a load function, $g_N = g_N(x,y)$ a Neumann boundary condition and $c \in \mathbb{R}^+$ is some constant.

Remark 2.1 (Model problem)**.** *The reaction diffusion problem is chosen for its inclusion of a zeroth order term. As will be demonstrated later, the resulting finite element system will contain two different element matrices allowing for a subsequent testing of both matrices.*

Remark 2.2 (Boundary conditions)**.** *As will be clarified later, the method proposed in this study mainly relates to the boundary of the domain. Neumann boundary conditions are used for this reason, in contrast to Dirichlet boundary conditions which are also common. This is because the Dirichlet condition imposes a direct constraint on the solution ($u = g_D$), whilst the Neumann condition only imposes a constraint on the flux, allowing for a study of the boundary with less influence from the boundary conditions.*

The first step onto obtaining the linear system stems from a reformulation of the problem to the *variational form* or *weak form*. This is carried out by multiplying the problem with a test function, v and integrating over the domain. After applying Green's theorem and the boundary conditions, we obtain

$$\int_\Omega \nabla u \cdot \nabla v dx dy + c\int_\Omega uv dx dy = \int_\Omega fv dx dy + \int_\Gamma g_N v dx dy. \tag{2.3}$$

The boundary conditions are weakly imposed, in contrary to conventional FEM where they are commonly imposed strongly (c.f. [14]). For the sake of the test function v behaving smoothly, we must introduce a suitable vector space. To do so, we introduce some notation.

2.1.1 Some introductory notation

Definition 2.1. Lebesgue spaces on a domain Ω are defined

$$L^p(\Omega) = \{v : \Omega \to \mathbb{R} : \|v\|_{L^p(\Omega)} < \infty\} \tag{2.4}$$

with the norm

$$\|u\|_{L^p} = \begin{cases} \left(\int_\Omega |u|^p dx dy\right)^{1/p}, & \text{if } 1 \le p < \infty \\ \sup\limits_{x\in\Omega} |u(x)|, & \text{if } p = \infty \end{cases} \tag{2.5}$$

Remark 2.3. *Lebesque spaces are commonly referred to as L^p-spaces.*

Definition 2.2. For $u \in L^p(\Omega)$ the weak derivative D^α is $g \in L^p(\Omega)$ as defined by

$$\int_\Omega g\varphi dx dy = (-1)^{|\alpha|}\int_\Omega u(D^\alpha \varphi) dx dy, \quad \forall \varphi \in D(\Omega), \tag{2.6}$$

where $D(\Omega)$ is defined

$$D(\Omega) = \{\varphi \in C^\infty(\Omega) : \text{supp} \subset\subset \Omega\}. \tag{2.7}$$

Definition 2.3. The Sobolev space on a domain Ω defined

$$W_k^p(\Omega) = \{u \in L^p(\Omega) : \|u\|_{W_k^p} < \infty\} \tag{2.8}$$

for L^p a Lebesgue space and the Sobolev norm defined such that

$$\|u\|_{W_k^p} = \begin{cases} \left(\sum_{|\alpha| \le k} \|D^\alpha u\|^p_{L^p(\Omega)} \right)^{1/p}, & \text{if } 1 \le p < \infty \\ \max_{|\alpha| \le k} \|D^\alpha u\|_{L^\infty(\Omega)}, & \text{if } p = \infty \end{cases} \tag{2.9}$$

where D^α is the weak derivative.

2.1.2 An abstract variational formulation

For Sobolev spaces, the case of $p = 2$ is also a Hilbert space for we obtain the L^2-inner product. This space is commonly used in FEM for its suitability with the test function, and it has attained its own notation defined $H^k(\Omega) := W_k^2(\Omega)$. Further, to attain a well-behaved solution for the problem defined, we must ensure that the functions up to the first order derivative are bounded. Thus, we choose $k = 1$ and get the space

$$H^1(\Omega) = \{u \in L^2(\Omega) : (\|u\|^2_{L^2(\Omega)} + \|\nabla u\|^2_{L^2(\Omega)})^{1/2} < \infty\} \tag{2.10}$$

which is equipped with the inner product $(\cdot,\cdot)$ and norm $\|\cdot\|$ defined as

$$\begin{aligned} (u,v)_{H^1(\Omega)} &= \int_\Omega uv dxdy + \int_\Omega \nabla u \cdot \nabla v dxdy = (u,v)_{L^2(\Omega)} + (\nabla u, \nabla v)_{L^2(\Omega)} \\ \|u\|_{H^1(\Omega)} &= (\|u\|^2_{L^2(\Omega)} + \|\nabla u\|^2_{L^2(\Omega)})^{1/2} \end{aligned} \tag{2.11}$$

on the domain. This imposes the condition that $u, v \in H^1(\Omega)$.

We proceed to write eq. 2.3 in an abstract formulation. Doing so, we introduce the linear and bilinear form. These are defined as follows.

Definition 2.4 (Linear form)**.** A linear form $l(\cdot)$ is a mapping $H^1(\Omega) \to \mathbb{R}$ such that for any $u, v \in H^1(\Omega)$

1. $l(u+v) = l(u) + l(v)$, and
2. $l(\alpha v) = \alpha l(v)$ for $\alpha \in \mathbb{R}$.

Definition 2.5 (Bilinear form)**.** A bilinear form $a(\cdot,\cdot)$ is a mapping $H^1(\Omega) \times H^1(\Omega) \to \mathbb{R}$ such that for any $u, v, w \in H^1(\Omega)$

1. $a(u+w, v) = a(u,v) + a(w,v)$,
2. $a(\alpha u, v) = \alpha a(u,v)$ for $\alpha \in \mathbb{R}$.

Stemming from the Lax-Milgram lemma, some interesting properties follow for the abstract variational problem (AVP).

Theorem 2.1. *If $a(\cdot,\cdot)$ is a continuous, coercive bilinear form and $l(\cdot)$ is a continuous linear form on $H^1(\Omega)$ then the abstract variational problem: find $u \in H^1(\Omega)$ such that*

$$a(u,v) = l(v), \quad \forall v \in H^1(\Omega) \tag{2.12}$$

has a unique solution u.

The interested reader is advised to [14] for details on the proof.

The abstract variational formulation for eq. 2.3 is then: find $u \in H^1(\Omega)$ for

$$a(u,v) = l(v), \quad \forall v \in H^1(\Omega) \tag{2.13}$$

such that

$$a(u,v) = (\nabla u, \nabla v)_{L^2(\Omega)} + (cu, v)_{L^2(\Omega)} \tag{2.14}$$

$$l(v) = (f, v)_{L^2(\Omega)} + (g_N, v)_{L^2(\Gamma)} \tag{2.15}$$

and to show existence and uniqueness of a solution we investigate the properties associated to theorem 2.1. First, we show coercivity for the bilinear form. We do this by showing that there $\exists \alpha > 0, \alpha \in \mathbb{R}$ such that

$$a(u,u) \geq \alpha \|u\|^2_{H^1(\Omega)}. \tag{2.16}$$

Defining $c_0 > 0, c_0 \in \mathbb{R}$ as a lower bound on on the constant c, the coercivity follows from

$$\begin{aligned} a(u,u) &= (\nabla u, \nabla u)_{L^2(\Omega)} + (cu, u)_{L^2(\Omega)} \geq \|\nabla u\|^2_{L^2(\Omega)} + c_0 \|u\|^2_{L^2(\Omega)} \\ &\geq \min(1, c_0)(\|\nabla u\|^2_{L^2(\Omega)} + \|u\|^2_{L^2(\Omega)}) \geq \alpha \|u\|^2_{H^1(\Omega)} \end{aligned} \tag{2.17}$$

with $\alpha = \min(1, c_0)$.

Now we shall show that the continuity of the bilinear form holds by showing there exists a $C_1 \in \mathbb{R}$ such that

$$|a(u,v)| = C_1 \|u\|_{H^1(\Omega)} \|v\|_{H^1(\Omega)} \tag{2.18}$$

which follows from

$$\begin{aligned} a(u,v) &\leq (\nabla u, \nabla v)_{L^2(\Omega)} + (cu, v)_{L^2(\Omega)} \\ &\leq \|\nabla u\|_{L^2(\Omega)} \|\nabla v\|_{L^2(\Omega)} + \|c\| \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \\ &\leq C_1 (\|\nabla u\|_{L^2(\Omega)} \|\nabla v\|_{L^2(\Omega)} + \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)}) \\ &\leq C_1 (\|\nabla u\|^2_{L^2(\Omega)} + \|u\|^2_{L^2(\Omega)})^{1/2} (\|\nabla v\|^2_{L^2(\Omega)} + \|v\|^2_{L^2(\Omega)})^{1/2} \\ &\leq C_1 (\|u\|_{H^1(\Omega)} \|v\|_{H^1(\Omega)}) \end{aligned} \tag{2.19}$$

where the Cauchy-Schwarz inequality has been applied twice.

Lastly, we shall show the continuity for the linear form by showing that there exists a $C_2 \in \mathbb{R}$ such that

$$|l(v)| = C_2 \|v\|_{H^1(\Omega)}. \tag{2.20}$$

We partition the terms into two parts

$$l(v) = (f, v)_{L^2(\Omega)} + (g_N, v)_{L^2(\Gamma)} = (1) + (2) \tag{2.21}$$

and obtain the following from (1)

$$(f, v)_{L^2(\Omega)} \leq \|f\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \leq C_3 \|\nabla v\|_{H^1(\Omega)} \tag{2.22}$$

as well as the following from (2)

$$(g_N, v)_{L^2(\Gamma)} \leq \|g_N\|_{L^2(\Gamma)} \|v\|_{L^2(\Gamma)} \leq C_4 \|v\|_{H^1(\Gamma)} \tag{2.23}$$

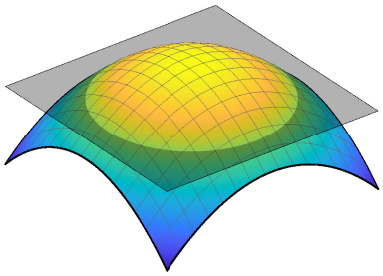
using Trace's inequality and thus finally arriving at

$$l(v) = (f, v)_{L^2(\Omega)} + (g_N, v)_{L^2(\Gamma)} \leq C_3 \|v\|_{H^1(\Omega)} + C_4 \|v\|_{H^1(\Gamma)} \leq C_2 \|v\|_{H^1(\Omega)} \tag{2.24}$$

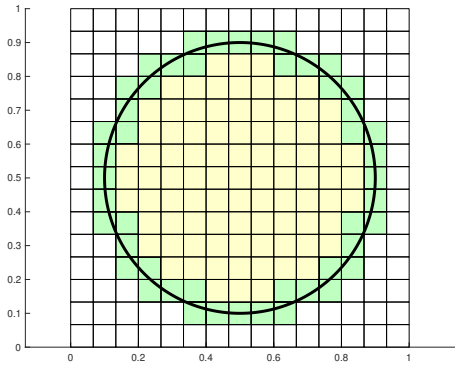
where $C_2 = \max(C_3, C_4)$, $C_3 = \|f\|_{L^2(\Omega)}$ and $C_4 = \|g_N\|_{L^2(\Gamma)}$.

2.2 The background mesh

As previously outlined, the major difference between conventional FEM and CutFEM lies within the mesh.Another common difference concerns the domain representation. In CutFEM, the boundary is often implicitly defined and instead of fitting the mesh to the domain, the domain is laid upon the mesh such that the boundary will cut through the elements underneath. One example of this is illustrated in fig. 2.1 for a circular domain, where the green region highlights the elements which are cut by the domain.



(a) Level set in $\mathbb{R}^3$ for circular representation in $\mathbb{R}^2$.



(b) Level set representation in $\mathbb{R}^2$ for circular domain. Green colouring shows cut elements, yellow colouring shows interior elements.

Figure 2.1: Level set representation of circular domain with representation in $\mathbb{R}^2$ for a quadrilateral mesh.

The domain representation can be done in a number of ways, where one common approach is to let the domain boundary be represented by a level set function. This is efficient for domains

which can be expressed using some function $\phi(\mathbf{x}, t)$, where $\mathbf{x}$ denotes the spatial coordinates and t denotes the temporal counterpart. The level set representation is especially convenient for cases with an evolving boundary, for the change on the boundary can be modelled over time.

In the two-dimensional and time-independent case, as studied in this work, the level set function is reduced to $\phi(x, y)$. If we denote the interior domain as $\Omega_1 \subset \Omega$, the exterior domain, Ω_2 and the common boundary as Γ for which $\Gamma \subset \Omega$, then it holds that

$$\phi(x,y)\begin{cases} > 0 & \Leftrightarrow (x,y) \in \Omega_1 \\ = 0 & \Leftrightarrow (x,y) \in \Gamma \\ < 0 & \Leftrightarrow (x,y) \in \Omega_2 \end{cases} \tag{2.25}$$

for all variations of function ϕ.

The background mesh is in this study chosen to consist of uniform quadrilateral elements, T_h, where the mesh size is determined by a parameter h. We denote the size dependent background mesh as $\mathcal{T}_{h,0}$ which allows us to define the active mesh, that is, the union of green and yellow elements as illustrated in fig. 2.1. The active mesh is then defined

$$\mathcal{T}_h = \{T_h \in \mathcal{T}_{h,0} : T_h \cap \Omega \neq \emptyset\}. \tag{2.26}$$

For easy referral later in this thesis, we also define the exterior $\mathcal{T}_{h,E}$, the interior $\mathcal{T}_{h,I}$ and the cut mesh $\mathcal{T}_{h,C}$ as

$$\mathcal{T}_{h,E} = \{T_h \in \mathcal{T}_{h,0} : T_h \cap \Omega_2 \neq \emptyset : T_h \cap \Omega = \emptyset\} \tag{2.27}$$

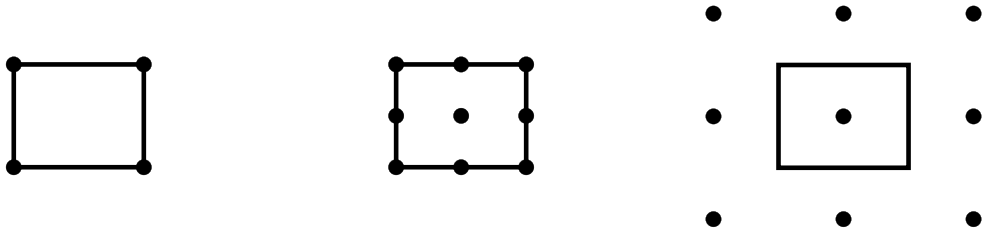
$$\mathcal{T}_{h,I} = \{T_h \in \mathcal{T}_{h,0} : T_h \cap \Omega_1 \neq \emptyset : T_h \cap \Omega_2 = \emptyset\} \tag{2.28}$$

$$\mathcal{T}_{h,C} = \{T_h \in \mathcal{T}_{h,0} : T_h \cap \Gamma \neq \emptyset\} \tag{2.29}$$

where the first is illustrated by the white elements in fig 2.1, the second is illustrated by the yellow elements and the last is illustrated by the green elements.

2.2.1 The use of C^1-splines

Our abstract variational formulation in eq. 2.13-2.15 is continuous on the space $H^1(\Omega)$. By imposing our mesh on the system, we wish to discretise the space by creating a finite element space $V_h \subset H^1(\Omega)$. This allows us to define the mesh in terms of elements with nodes equipped with basis functions. An example Lagrange element with nodal placements corresponding to polynomial basis functions of order 1 and 2 as well as an example element for C^1-spline basis functions of order 2 is illustrated in fig. 2.2. The second order is convenient for including rounded boundary representations, whereas the first order represents each segment as a linear, first order approximation. For quadrilateral elements, the first order functions induce four nodal points per element, whilst the second order functions induce nine nodal points. The spline elements get contributions from nodes with location outside of the element due to the overlap from nearby spline basis functions.



(a) Lagrange order 1. (b) Lagrange order 2. (c) Spline order 2.

Figure 2.2: Elemental nodes for a first and second order quadrilateral element with polynomial and spline basis functions.

A finite element function is expressed as a linear combination of all basis functions, and locally, on a single element such a function will be a polynomial. For second order spline basis functions this polynomial space is $Q^2 = \text{span}\{1, x, y, xy, x^2, y^2, x^2y, xy^2, x^2y^2\}$ and hence, any function $v \in Q^2$ can be expanded

$$v = a^{(1)}\varphi_1 + a^{(2)}\varphi_2 + ... + a^{(8)}\varphi_8 + a^{(9)}\varphi_9 = \sum_{k=1}^{9} a^{(k)}\varphi_k \tag{2.30}$$

with basis functions φ_i and $i = 1,..,9$. Since derivatives of a function in Q^2 also belong to (a subspace of) Q^2, we can locally express derivatives of basis functions in terms of the basis functions themselves

$$\frac{\partial \varphi_i}{\partial x} = \sum_{k=1}^{9} b_i^{(k)}\varphi_k, \quad \frac{\partial \varphi_i}{\partial y} = \sum_{k=1}^{9} c_i^{(k)}\varphi_k. \tag{2.31}$$

The coefficients can be found by solving the system.

In this work, spline functions as shown in above illustration, called C^1-splines of second order, are used in $\mathbb{R}^2$. The specific spline functions in two dimensions become a tensor product as given by

$$\varphi_k = B^2_{k,h}(x,y) = B^2(x/h - k_x)B^2(y/h - k_y), \quad k_x, k_y \in \mathbb{Z} \tag{2.32}$$

for a uniform mesh with mesh size h. The dimension specific splines can be determined from the Cox-de-Boor recursion formula, stating

$$B_i^0(x) = \begin{cases} 1 \text{ if } t_i \le x \le t_{i+1}, \\ 0 \text{ otherwise} \end{cases} \tag{2.33}$$

$$B_i^p(x) = \frac{x - t_i}{t_{i+p} - t_i} B_i^{p-1}(x) + \frac{t_{i+p+1} - x}{t_{i+p+1} - t_{i+1}} B_{i+1}^{p-1}(x) \tag{2.34}$$

where t_i for $i = 1,...,n$ denotes some discretisation of parameter x, n denotes the number of partitions and $p = 1,2$. This yields the following corresponding expansion of a function v within the space of second order C^1-splines on $\mathbb{R}^2$

$$v = \sum_{k=1}^{9} a^{(k)}\varphi_k = \sum_{k=1}^{9} a^{(k)} B^2_{k,h}(x,y) \tag{2.35}$$

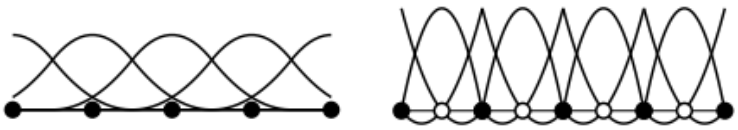
and the result from eq. 2.31 follows directly. For further information on spline functions in FEM, see [1, 15].

The choice of C^1-splines in the level set representation of the domain has two useful properties.

1. Additional smoothness of the boundary, limiting the set of admissible geometries to more sane shapes compared to Q^2 Lagrange basis functions.

2. The C^1-splines are non-negative whilst the Q^2 polynomials have some negative contribution, see fig. 2.3 for illustration.

The latter contribute to the level set values. For the spline basis functions it means that element-wise paired nodes with all positive or negative values can be determined as an exterior or interior element respectively, whilst there may occur some non-trivial variations for the polynomial case. The predictability of the elemental level set values is useful for an efficient element classification, which will be discussed further later. However, due to the great overlaps of supporting splines to a node, the nodes in close proximity to the boundary attain some variations which introduces some uncertainties in the elements containing both negative and positive level set values. This is conventionally solved by applying some quadrature rule to include or outrule any element in the active mesh and will be discussed further later.



(a) C^1-splines. (b) Polynomial.

Figure 2.3: Second order nodal functions in $\mathbb{R}^1$ [1].

The resulting finite element space for C^1-splines thus becomes

$$V_h = \{v : v \in C^1(\Omega), v|_{T_h} \in Q^2, \forall T_h \in \mathcal{T}_h\} \tag{2.36}$$

and is thus defined on the active mesh.

2.2.2 Mapping to the reference element

It is common to make use of an isoparametric mapping in finite element methods to allow for computation of curved elements by mapping them from a higher order element to a first order element type (more on this can be found in [14]). An isoparametric map like such is not used in this work, however, a linear map from an actual element to a reference element can be useful. The reason for this will be slightly touched upon in the upcoming section, and it will be further clarified in section 3.3.2 why this is especially useful.

For some quadrilateral element K with dimensions h_x, h_y, a suitable reference element may be the unit square $\hat{K} = [0,1] \times [0,1]$. For the reference coordinates $(\hat{x}, \hat{y})$ we can define the map $p(\hat{x}, \hat{y}) : \hat{K} \rightarrow K$ between the elements with the corresponding Jacobian

$$J = \begin{bmatrix} \frac{\partial \hat{x}}{\partial x} & \frac{\partial \hat{y}}{\partial x} \\ \frac{\partial \hat{x}}{\partial y} & \frac{\partial \hat{y}}{\partial y} \end{bmatrix} = \begin{bmatrix} h_x & \\ & h_y \end{bmatrix}. \tag{2.37}$$

. The basis functions which have been used are then related to the corresponding reference by

$$\varphi_i = \hat{\varphi}_i \circ p = \hat{\varphi}_i(p(\hat{x}, \hat{y})) \tag{2.38}$$

with gradient

$$\nabla \varphi_i = J^{-T} \hat{\nabla} \hat{\varphi}_i = \begin{bmatrix} h_x^{-1} & \\ & h_y^{-1} \end{bmatrix} \begin{bmatrix} \frac{\partial \hat{\varphi}_i}{\partial \hat{x}} \\ \frac{\partial \hat{\varphi}_i}{\partial \hat{y}} \end{bmatrix} \tag{2.39}$$

and the previous results thus directly translates with above.

2.3 A finite element formulation of the model problem

With the abstract variational problem as obtained in eq. 2.13-2.15 and the finite element vector space V_h as defined in eq. 2.36, we can now define the finite element formulation as: find $u_h \in V_h$ for

$$a_h(u_h, v) = l_h(v), \quad \forall v \in V_h \tag{2.40}$$

such that

$$a_h(u_h, v) = (\nabla u_h, \nabla v)_{L^2(\Omega)} + (c u_h, v)_{L^2(\Omega)} \tag{2.41}$$

$$l_h(v) = (f, v)_{L^2(\Omega)} + (g_N, v)_{L^2(\Gamma)}. \tag{2.42}$$

Remark 2.4. *Note that a stabilisation term should be included in the finite element formulation to ensure that the cut elements obey certain properties. This will be further discussed in section 2.3.2, and the term is therefore excluded until then.*

2.3.1 The element matrices

Now that we have the finite element formulation of the problem as well as our basis functions, we can formulate the expression in terms of these basis functions. We have that $V_h = \text{span}(\{\varphi\}_{i=1}^N)$, where N is the degrees of freedom, and as previously discussed, any function $v \in V_h$ can be expressed using a combination of the basis functions. For the solution u_h and the test function v, this gives us

$$u_h = \sum_{i=1}^{N} \xi_i \varphi_i, \quad v = \sum_{j=1}^{N} \varphi_j \tag{2.43}$$

which can be written into the system in eq. 2.40-2.42. For the bilinear form in eq. 2.41, this yields

$$\begin{aligned} a_h(u_h, v) &= \int_\Omega \nabla u_h \cdot \nabla v dxdy + c\int_\Omega u_h v dxdy = \\ &= \int_\Omega \nabla\left(\sum_{i=1}^{N} \xi_i \varphi_i\right) \cdot \nabla\left(\sum_{j=1}^{N} \varphi_j\right) dxdy + c\int_\Omega \left(\sum_{i=1}^{N} \xi_i \varphi_i\right)\left(\sum_{j=1}^{N} \varphi_j\right) dxdy \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \xi_i \left(\int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j dxdy + c\int_\Omega \varphi_i\varphi_j dxdy\right) \end{aligned} \tag{2.44}$$

and for the linear form in eq. 2.42 we obtain

$$\begin{aligned} l(v) &= \int_\Omega f v dxdy + \int_\Gamma g_N v dxdy = \int_\Omega f\sum_{j=1}^{N} \varphi_j dxdy + \int_\Gamma g_N \sum_{j=1}^{N} \varphi_j dxdy \\ &= \sum_{j=1}^{N} \int_\Omega f\varphi_j dxdy + \int_\Gamma g_N \varphi_j dxdy \end{aligned} \tag{2.45}$$

which we can assemble into a linear system. This linear system gives us

$$(A + cM)\xi = b + n, \quad \text{with} \tag{2.46}$$

$$\begin{aligned} A_{ij} &= \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j dxdy, \quad M_{ij} = \int_\Omega \varphi_i\varphi_j dxdy \\ b_j &= \int_\Omega f\varphi_j dxdy, \quad n_j = \int_\Gamma g_N\varphi_j dxdy \end{aligned} \tag{2.47}$$

and we have thus obtained our stiffness matrix A, mass matrix M, load vector b and Neumann boundary condition vector n.

As we discussed in section 2.2.2, we ought to perform a mapping from an element K to $\hat{K}$. To do this, we must study the element-wise entries of the element matrices. We retrieve the following by using eq. 2.38-2.39

$$\begin{aligned} A_{ij}^K &= \int_{\Omega\cap K} \nabla\varphi_i \cdot \nabla\varphi_j dxdy = \int_{\Omega\cap\hat{K}} (J^{-T}\hat{\nabla}\hat{\varphi}_i)^T J^{-T}\hat{\nabla}\hat{\varphi}_j \det(J) d\hat{x}d\hat{y} \\ M_{ij}^K &= \int_{\Omega\cap K} \varphi_i\varphi_j dxdy = \int_{\Omega\cap\hat{K}} \hat{\varphi}_i\hat{\varphi}_j \det(J) d\hat{x}d\hat{y} \end{aligned} \tag{2.48}$$

and the same method is applied to right and side, $l(v)$. We note that $\det(J) = h_x h_y$ and retrieve an interesting relationship for the mass matrices. We get

$$M_{ij}^K = h_x h_y \int_{\Omega\cap\hat{K}} \hat{\varphi}_i\hat{\varphi}_j d\hat{x}d\hat{y} = h_x h_y M_{ij}^{\hat{K}} \tag{2.49}$$

and we thus have a direct scaling between the element K and the reference element $\hat{K}$. From our previous discussion regarding the relation between the basis functions and their derivatives, as shown in eq. 2.31, we can extend this to the reference element and re-write the expression of

the stiffness matrix. We get

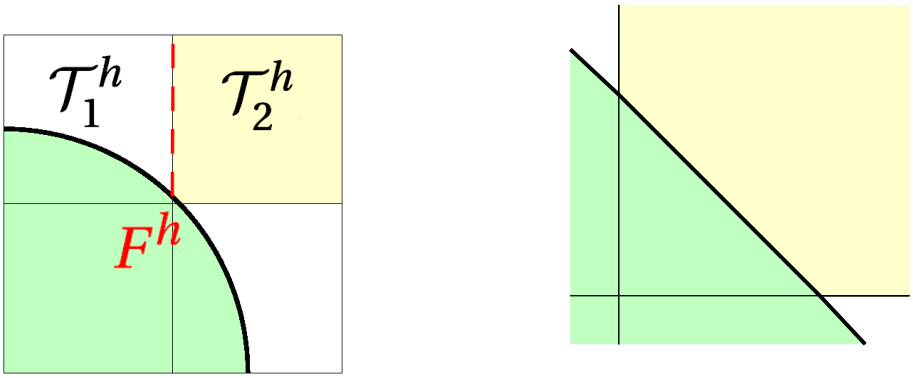
$$\begin{aligned} A_{ij}^K &= h_x h_y \int_{\Omega\cap\hat{K}} (J^{-T}\hat{\nabla}\hat{\varphi}_i)^T J^{-T}\hat{\nabla}\hat{\varphi}_j \, d\hat{x}d\hat{y} \\ &= \frac{h_y}{h_x}\int_{\Omega\cap\hat{K}} \frac{\partial\hat{\varphi}_i}{\partial\hat{x}}\frac{\partial\hat{\varphi}_j}{\partial\hat{x}} d\hat{x}d\hat{y} + \frac{h_x}{h_y}\int_{\Omega\cap\hat{K}} \frac{\partial\hat{\varphi}_i}{\partial\hat{y}}\frac{\partial\hat{\varphi}_j}{\partial\hat{y}} d\hat{x}d\hat{y} \\ &= \frac{h_y}{h_x}\int_{\Omega\cap\hat{K}} \sum_{k,l=1}^{p} b_i^{(k)}\hat{\varphi}_k\hat{\varphi}_l b_j^{(l)} \, d\hat{x}d\hat{y} + \frac{h_x}{h_y}\int_{\Omega\cap\hat{K}} \sum_{k,l=1}^{p} c_i^{(k)}\hat{\varphi}_k\hat{\varphi}_l c_j^{(l)} \, d\hat{x}d\hat{y} \\ &= \frac{h_y}{h_x} b_i^T M^{\hat{K}} b_j + \frac{h_x}{h_y} c_i^T M^{\hat{K}} c_j \end{aligned} \tag{2.50}$$

and can thus assemble the local stiffness matrix from the local reference mass matrix if we previously have computed the coefficients in eq. 2.31. For a mesh with the same reference element for all elements, this only has to be performed once, for the results will hold for all elements.

Remark 2.5. *The load vector can also be assembled in terms of the mass matrix, albeit the global one. This is achieved by performing an L^2-projection of the load function f onto V_h, yielding the discrete $P_h f \in V_h$. The load vector b is then obtained from $b = MP_h f$. This will prove useful later.*

2.3.2 Ghost penalty

Not fitting the mesh to the domain may result in a large variation of accumulated intersections between an element and the domain. Some of the partially intersecting elements will almost entirely be covered by the domain, whilst some elements will have a very small intersection area. One example of an element with little domain intersection is illustrated in fig. 2.4. This type of intersection produces low valued matrix entries after integration, resulting in a poor conditioning of the element matrices which may be fatal for the linear system.



(a) Four elements on domain.

(b) Closer inspection of slightly intersecting element.

Figure 2.4: Elements on geometry with instance of small domain intersection as highlighted by yellow.

This can be overcome by extending the coercivity from the domain of the system, as was shown to hold in section 2.1.2, to all of the elements intersecting the domain. This was proposed in [6] by introducing a penalty term, *ghost penalty*, and some further notes on the stabilisation is also discussed in [16, 3]. The jump in derivative between the elements must be accounted for to form this fictious domain. We denote one of the elements $\mathcal{T}_1^h$ and the other $\mathcal{T}_2^h$, introducing a separating facet F^h in a set of all facets $\mathcal{F}$ (see fig. 2.4). the jump in derivative over the facet can be denoted by a jump operator as defined by

$$[\![u]\!] = u_1|_{F^h} - u_2|_{F^h} \tag{2.51}$$

for $u_1 \in \mathcal{T}_1^h$ and $u_2 \in \mathcal{T}_2^h$.

This allows us to introduce the penalty term as given by

$$j_h(u_h, v) = \sum_{F^h \in \mathcal{F}} \sum_{l=1}^{k} (\gamma h^{2l+1} [\![D_{\mathbf{n}}^l u_h]\!], [\![D_{\mathbf{n}}^l v]\!])_{F^h} \tag{2.52}$$

with $D_{\mathbf{n}}^l$ is a normal partial derivative to the element facets (c.f. [4]), γ being a positive penalty parameter which can be set to $\gamma = 1$ [16] and k is the elemental order, which is $k = 2$ in this study.

Remark 2.6. *The first order derivative provides no contribution since this study is performed using C^1-splines. Therefore the second summation can be removed and we can set $l = k = 2$ for any further uses of the penalty term.*

The final finite element formulation with ghost penalty term specific for this study is given by

$$A_h(u_h, v) = l_h(v), \quad \forall v \in V_h \tag{2.53}$$

such that

$$A_h(u_h, v) = a_h(u_h, v) + j_h(u_h, v) \tag{2.54}$$

$$\tag{2.55}$$

with the terms as given by

$$a_h(u_h, v) = (\nabla u_h, \nabla v)_\Omega + (c u_h, v)_\Omega \tag{2.56}$$

$$l(v) = (f, v)_\Omega + (g_N, v)_\Gamma \tag{2.57}$$

$$j_h(u_h, v) = \sum_{F^h \in \mathcal{F}} (h^5 [\![D_{\mathbf{n}}^2 u_h]\!], [\![D_{\mathbf{n}}^2 v]\!])_{F^h} \tag{2.58}$$

and our finite element formulation is complete.

2.3.3 A priori error estimate for CutFEM

For the analytical solution $u \in H^1(\Omega)$ and the cut finite element solution $u_h \in V_h$ as derived from the finite element problem defined in eq. 2.53-2.58 a priori error estimate can be derived. This

is directly dependent on the order of the finite element space, which is as previously defined 2. It then holds that the optimal error bound on the L^2-norm error on the domain Ω is given by

$$\|u - u_h\|_{L^2(\Omega)} \lesssim h^3 \tag{2.59}$$

and the optimal error bound on the energy sub-norm on the domain Ω is given by

$$|||u - u_h||| = \|\nabla(u - u_h)\|_{L^2(\Omega)} \lesssim h^2 \tag{2.60}$$

where $\lesssim$ denotes less than or equal to a constant $C \in \mathbb{R}$. Above follows from usage of the Galerkin orthogonality and Cea's lemma.

CHAPTER 3

THINKING OF NEURAL NETWORKS

Artificial intelligence, and especially machine learning and neural networks, is becoming increasingly popular in data science. We may not yet fully understand our own brains, however, by mimicking the known behaviours of our neurons, neural networks offer promising results regarding pattern recognition in complex data sets. The multi-tasking properties of the networks allow for a relatively quick evaluation of high-dimensionality data, making it especially relevant for improving the efficiency of unfitted finite element methods.

In this chapter, the motivations for the neural networks of choice are outlined, as well as the pre- and post-handling of the data. Moving forward, the handling of the data will altogether be referred to as *filtering*. Lastly, some architectural benchmarks are discussed.

3.1 The essentials of a CutFEM quadrature rule

The elemental matrices as shown in eq. 2.48 are relatively easy evaluated in a conventional finite element method, since the intersection with the domain occurs over the entire element. For a uniform mesh and a sufficiently regular problem, all local matrices look the same and the matrices can thus be computed beforehand. For the case of CutFEM, this still holds for the interior elements, however, the cut elements must be computed using quadrature of increased complexity. This occurs due to the intersection between a specific element and domain, $K \cap \Omega$, and as seen in eq. 2.48 this intersection must be integrated over to obtain the element matrices. This is performed using a sub-discretisation and Gaussian quadrature in [3], whilst [9] implements a dimensionality reduction of the initial volume integrals. In [10] a number of integration methods are investigated and compared, although most of the methods are deemed to be insufficient for the task. The difficulty of integration stems from an increased order of the elements, for this quickly makes the integration process cumbersome.

It was mentioned briefly in section 2.2.1 that the overlap of spline basis functions results in certain elements which are difficult to classify from their sets of level set values. As said, this is

usually solved by applying quadrature on all elements, then outruling the elements with empty support and using the resulting active elements in the assembly of the element matrices.

With above as motivation, removing the quadrature must not only replace the process of integration, but it must also determine whether the element in question provides any global contribution to the linear system. This can be done by classifying the elements based on some grounds of choice on beforehand and thus only apply the integration process over the cut elements (as mentioned, the interior elements can be pre-computed). This would be ideal since, given it can be implemented sufficiently effective, it would remove the requirement for quadrature of interior and exterior elements. In this context, *sufficiently efficient* must be related to the time at which the quadrature currently takes and the resulting precision.

Further, since the quadrature ought to be replaced by some method based upon a neural network, specifically it must be a regression network, it would be an advantage to only allow the elements that truly are to be integrated pass through the network. Otherwise, the network would have to act as a regression network and a classification network subsequently, introducing the risk of over-fitting or merely a reduced performance in a mean error sense by including these data points.

With above as motivation, the preluding classification task can be performed using a classification network, which labels the elements before they are given to the regression network, which performs an approximation of the integration as displayed in eq. 2.48. This is presented in a brute force pseudo algorithm in 1, for easy overview of each specific step. *Rnet* denotes the regression network, and *Cnet* denotes the classification network.

Algorithm 1 CutFEM using neural networks (brute force)

$\mathcal{T}_{h,0} \leftarrow$ background mesh
$ls \leftarrow$element-wise level set values
$N \leftarrow$ number of elements
$F \leftarrow$ filtering functions
for $k = 1 : N$ **do**
    $\widetilde{ls}(k) \leftarrow F(ls(k))$
    Class $\leftarrow$ Cnet($\widetilde{ls}(k)$)
    if Class == Interior **then**
        $M^K \leftarrow$ pre-computed
    else if Class == Cut **then**
        $M^{\hat{K}} \leftarrow$ Rnet($\widetilde{ls}(k)$)
        $M^K \leftarrow F^{-1}(M^{\hat{K}})$
    else
        $M^K \leftarrow 0$
    end if
    $M \leftarrow M + M^K$
end for

The same method is expressed in a more efficient, compressed version in 2, where everything is computed in batch instead of element-wise.

Algorithm 2 CutFEM using neural networks (efficient)

$\mathcal{T}_{h,0} \leftarrow$ background mesh
$ls \leftarrow$element-wise level set values
$F \leftarrow$ filtering functions
$\widetilde{ls} \leftarrow F(ls)$
$\{\mathcal{T}_{h,E}, \mathcal{T}_{h,C}, \mathcal{T}_{h,I}\} \leftarrow \text{Cnet}(\widetilde{ls})$
$M^{\hat{K}} \leftarrow \text{Rnet}(\widetilde{ls}(\mathcal{T}_{h,C}))$
$M^K(\mathcal{T}_{h,C}) \leftarrow F^{-1}(M^{\hat{K}})$
$M^K(\mathcal{T}_{h,I}) \leftarrow$ pre-computed
$M \leftarrow$ assemble M^K

One advantage of the compressed version is that the outer elemental loop is removed. There must, of course, remain some traces of an element-wise loop in the filtering and assembly loops, however, comparing algorithm 1 and algorithm 2, each call to external functions only has to be performed once.

Remark 3.1. *The algorithms above only display the procedure of the mass matrix. This is because the stiffness matrix and the load vector can be derived using the mass matrix, see section 2.3.1 where the mathematical operations and constant values are discussed. The load vector is assembled using the global stiffness matrix, hence it is constructed once the above algorithms have been carried out. The stiffness matrix is assembled from the local reference mass matrix, and it thus has to be computed before the inverse filtering is applied. It is computed using eq. 2.50 and is assembled into a global correspondence similarly to the mass matrix.*

3.2 Production of the data set

The data set must be produced so that the training data is representative of the geometries that may occur for an arbitrary discretisation. For a sufficiently small mesh size, any domain-intersecting element may be approximated as a plane of some angle and orientation. However, as this imposes some major constraint on the mesh, we want to allow for some slack. In fig. 3.1, a slightly curved domain for a relatively coarse mesh size is illustrated. It is apparent from the example elements that most of the domain-intersecting elements can be approximated by either a plane or some part of a circular domain, such as a circle or a hole.

This motivates the choice to include planar and circular geometries, where the latter includes circles as well as holes. However, this excludes the occurrence of corner singularities, as investigated for CutFEM in for instance [17], and can instead be left as a potential extension of the method.

For these geometries to be a reasonable representation, it does pose some restrictions on the mesh size that can be implemented by the user. The mesh size must be chosen with the sharpest curvature of the domain in mind, making each intersection a reasonable approximation of a plane, circle or a hole. The mesh size is chosen to be uniform in this study, that is, $h = h_x = h_y$,

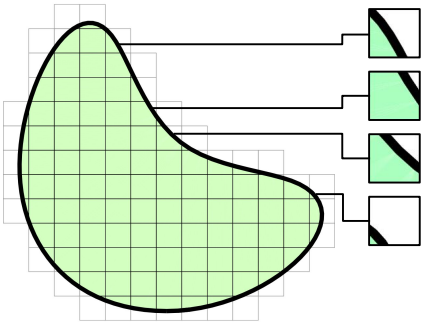


Figure 3.1: Curved *"bean"* domain with coarse mesh and highlighted examples of domain-intersecting elements.

hence any notation separating the mesh size in the spatial coordinates is dropped. Further, all geometries are generated upon the unit square, that is $[0,1] \times [0,1]$.

3.2.1 Circular geometries

The circular geometries consist of circles and holes. The general level set function for a circle is

$$\phi_c(x,y) = R^2 - (x-x_c)^2 - (y-y_c)^2 \tag{3.1}$$

and for a hole the corresponding function is defined

$$\phi_h(x,y) = -\phi_c(x,y) = (x-x_c)^2 + (y-y_c)^2 - R^2 \tag{3.2}$$

where x_c, y_c denotes the center coordinates and R denotes the radius. For all further discussion, only the circle will be considered, for the results of the hole are analogous.

To attain some variations in the intersections, the main idea is to generate circles of varying radius over a defined range $(0, 0.5) \approx [R_{min}, R_{max}]$, with a defined increment dR. By including some randomness to the center coordinates, further variations can be included. This is modelled by randomising the center coordinates within the range of half an element from the center $(0.5, 0.5)$ resulting in an allowed range of coordinates $x_c, y_c \in [0.5 - h/2, 0.5 + h/2]$. The values of x_c and y_c are determined from

$$\begin{gathered} x_c = 0.5 + \xi_x h, \quad y_c = 0.5 + \xi_y h \\ \xi_x, \xi_y \in [-0.5, 0.5] \end{gathered} \tag{3.3}$$

where ξ_x, ξ_y are stochastic, uniformly distributed variables.

It must also be considered that there is a greater variation between elements relating to a circle of smaller radius than a circle of larger radius. This must be taken into account for two reasons.

1. A smaller radius results in less elements actually intersecting with the domain, resulting in only a small fraction of the elements in the total data set having much curvature if not taken into accounts.

2. A smaller radius in relation to the mesh size results in more curvature of the intersecting domain, producing greater variations between any two elements, hence more elements of this kind must be included to represent all the varying types of cuts.

This motivates the introduction of zones in the discretisation. The zones represent different sizes of the radius increment dR and are demonstrated in fig. 3.2. The domain and mesh size are merely for illustration. The radius increment is chosen to be dependent on the mesh size and for the three zones it should hold

$$dR_r(h) < dR_b(h) < dR_g(h) \tag{3.4}$$

where the subscripts denotes the colour zones and is chosen in relation to the mesh size. r denotes the zone up to the red marking, b denotes the zone between the red and the blue marking and g denotes the zone between the blue and the green marking. By doing this, we ensure to represent the greater variation of the intersections of the smaller circles, and to not over represent the close to planar cuts as occur for the circles of larger radius.

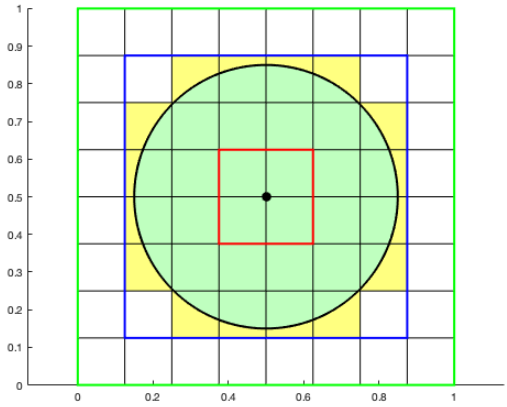


Figure 3.2: Example of circular domain with three zones representing different radius increments dR.

3.2.2 Planar geometries

The planar geometries are generally given by

$$\phi_p(x, y) = ax + bx - d \tag{3.5}$$

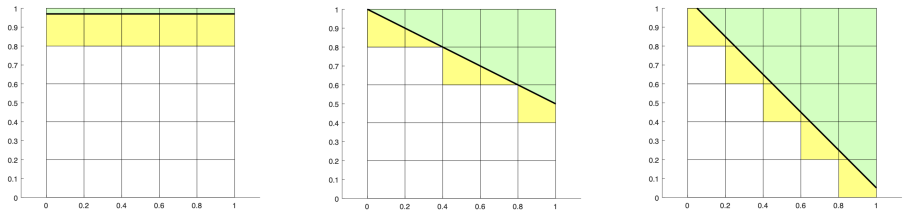
where $a, b, c \in [0, 1]$ are constants with tilt coefficients a, b and level of the plane d. Generally, three different planes may occur within the set of the cut elements. These types differ from each other in the incline θ of the plane. The three categories can be defined to

1. planes of $\theta = 0$ degree incline (straight),

2. planes of $0 < \theta < 45$ degrees incline (tilted), and

3. planes of $\theta = 45$ degrees incline (diagonal)

and are demonstrated in fig. 3.3. All geometries are created along one axis and one orientation due to the filtering processes which will be discussed later.



(a) Straight plane. (b) Tilted plane. (c) Diagonal plane.

Figure 3.3: The three types of variations of the planar cuts as included in the data set of this thesis.

The straight type of cuts as shown in fig. 3.3 and fig. 3.4 occur for $a = 0$ and $b = 1$ for different values of $d \in [0.5 - h/2, 0.5 + h/2]$. These need only be iterated over the closest elements to the middle of the background domain, since any further iterations will merely be repetitions of the same values thereafter.

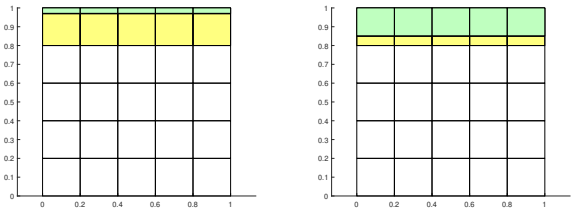


Figure 3.4: Illustration of straight plane at different cuts within one element.

The tilted type of cuts as shown in fig. 3.3 and fig. 3.5 occur for $a \in [h, 1-h]$, $b = 1$ and $d = 1$. The coefficient a must be iterated along the entire side to generate a sufficient representation of cuts. As discussed, due to symmetries and filtering, only one orientation need to be investigated.

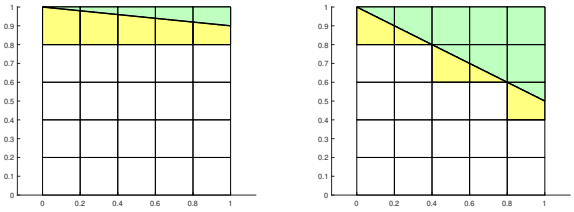


Figure 3.5: Illustration of tilted plane at different cuts along the reference element.

The diagonal type of cuts as shown in fig. 3.3 and fig. 3.6 occur for $a = 1$, $b = 1$ and $d \in [1 - h, 1 + h]$. Similar to the straight case, the cuts only need to be iterated over one element since any further iterations would result in repetitions of the same geometries.

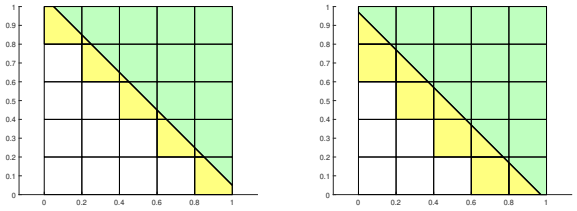


Figure 3.6: Illustration of tilted plane at different cuts along the reference element.

3.3 Filtering of the data

The geometries which has been motivated above also comes with some weaknesses which must be addressed, and when possible, remedied. The strengths and weaknesses with including circular geometries has already been mentioned and motivated for the intentions of this method, hence this will not be further discussed in this section. Instead, there are a few other points to look at which will motivate the choices of filtering methods for the data set of choice. These are as follows.

1. The level set values depend directly on the level set function of choice, and whilst it ideally will be near zero close to the boundary this will differ between different geometries. ”Near zero” is essentially a relative term depending on the magnitude of values any level set representation takes on in general. If, for instance, the level set function is relatively flat in three dimensions, this may lead to level set values which are particularly close to zero even for interior elements, and the network may be ill fitted to handle this type of input.

2. The intersection between the element and the domain is directly affected by the mesh size as the element matrices are integrated over the element. If the element is small, this will be represented in the set of local contributions. In the most simple case, this essentially means that a smaller element size produces a smaller intersection area.

3. The orientation of the cut element produces level set values in varying orders. For instance, if the intersection occurs in the upper right corner, the nodes related to the intersection will attain larger values. However, if the intersection is around the lower left corner different nodes will be associated with these values. This produces a large variation in the input resulting in unnecessary symmetries and potentially an increased dimensionality of the problem.

All these potential issues can be remedied to some extent, which will be outlined in the following sections.

3.3.1 Normalisation of level-set values

The first point concerns the variations of the set of level set values between different elements, and more importantly, different geometries. This can be overcome by normalisation and

performed on the entire training set to attain some representative variations in the data.

Because it is not straightforward whether a positive or negative node is inside or outside the boundary of the domain, one must instead look at the variations in the nearby nodes within the element to determine its probable position. Whilst this determination approach is mainly relevant for the classification network, the node-wise result does affect the numeric results of the regression network and hence it follows a similar reasoning. The requirement for comparison to nearby nodes introduces a relevance in comparing relative magnitudes within an element. For this reason, we look at a specific element-wise normalisation in this work.

The goal is to attain a pre-defined variation within each set of nine level set values. One may motivate a probabilistic interpretation for each node is directly or indirectly classified as interior or exterior, which implies that the range $[0,1]$ may be of interest. However, as we do not wish to lose the signum properties of the values, we may extend the range to $[-1,1]$.

We may denote the set of level set values of an element k as

$$LS^k = \{ls_1^k, ls_2^k, ls_3^k, ls_4^k, ls_5^k, ls_6^k, ls_7^k, ls_8^k, ls_9^k\} \tag{3.6}$$

for nine nodal points on the element and $ls_i^k \in \mathbb{R}$ for $i = 1,..,9$. The normalisation of the element is then reduced to a scaling with the largest value (in magnitude) of the values ls_i^k. Again, as we do not want to change the sign, we must scale with the absolute value. This yields a normalised set

$$\widetilde{LS}^k = \frac{LS^k}{|\max(LS^k)|} \tag{3.7}$$

for the element k. This will ensure that the largest element in magnitude will attain a value of 1 or -1, with the others ranging within $[-1,1]$.

3.3.2 Scaling to reference element

The second point concerns how the element matrices are affected by the mesh size. This can be solved in many ways, however, many which increases the dimensionality and input data of the problem rather than reducing it. One way which reduces the complexity is already outlined in a few sections in chapter 2. One way to represent an element or intersection with undesired geometry is by mapping the element K to a reference element $\hat{K}$. If the reference element is determined to be the unit square, all reference element matrices will be within an expected and uniform range regardless of mesh size. Training the network to approximate the local reference matrix is thus a better choice in terms of dimensionality. This simplification of the problem requires a step of invers-filtering to convert the local reference matrix to the true local matrix. However, looking at eq. 2.49, we see that the true local mass matrix can be obtained by direct scaling with the mesh size according to

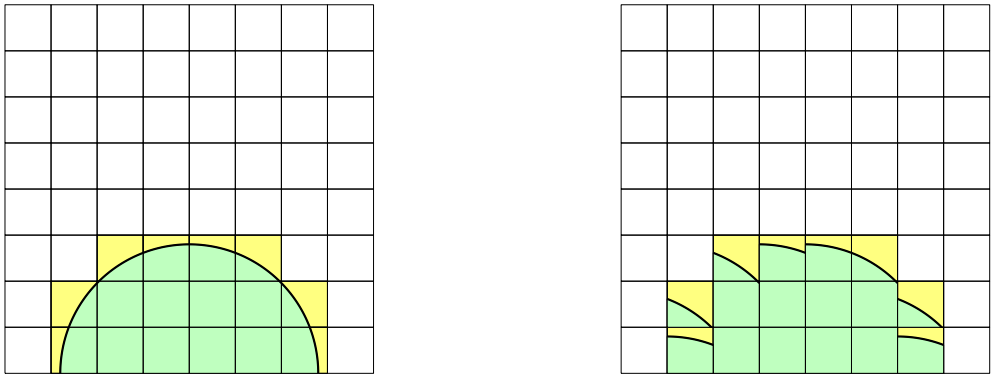
$$M^K = h_x h_y M^{\hat{K}} \tag{3.8}$$

where the values in $M^{\hat{K}}$ stems directly from the Rnet output. Further, looking at eq. 2.50, we are reminded that the stiffness matrix can be determined from the local reference mass matrix, and therefore both element matrices are quickly obtained from the reference element.

3.3.3 Permutation of level set values

The third point mentions the symmetries and the variations in the sequences of the magnitudes of level set values. It was also briefly indicated in section 3.2 that the any symmetries ought to be removed, hence the motivation as why to only include certain variations of the planes. An interchange in the sequences introduces an idea of a sorting of each set of elemental level set values, so that the highest values are first or last in the sequence. However, as this might result in a permutation of neighbouring nodal points, this would introduce new uncertainties and thus an other approach of sorting must be carried out instead.

Preferably, each element intersecting the domain would be orientated equally in terms of geometry. That is, the nodal points with the *heaviest* nodes (nodes with highest level set values) would be oriented along a pre-defined side or corner, which can be achieved by geometric transformations such as mirroring and rotation of the element. The idea is illustrated in fig. 3.7 where the heaviest weights are defined to be permutated to the lower, left corner and lower side.



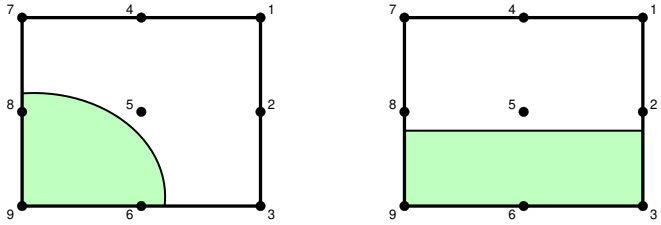
(a) Before permutation.

(b) After permutation.

Figure 3.7: Example of geometry before and after level set values have been permutated along the lower left corner and lower side.

This motivates partitioning the type of intersections into two types of geometries; *corner geometries* and *side geometries*. An example of these are illustrated, at desired nodal location, in fig. 3.8. The elemental nodes are numbered in both figures. A corner geometry may also include diagonal planar cuts, and side geometries may also include circles intersecting with a side of the element, thus, there may be many variations on the types of intersections included in the class.

Remark 3.2. *Note that a second order Lagrange element nodal distribution is used in the above and following figures to better illustrate the points. The differences between the Lagrange and the spline nodal positions are illustrated in fig. 2.2, yet due to the similar configuration the same transformations hold and the Lagrangian nodes are used here for illustrative purposes.*



(a) Corner geometry. (b) Side geometry.

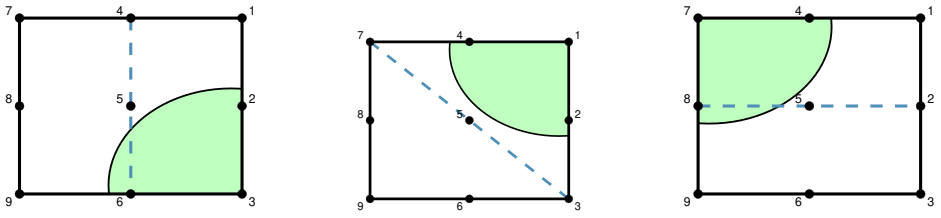
Figure 3.8: Example of corner and side geometry after transformation to desired nodal location.

Above essentially means that the *heaviest* (highest valued) level set values should be located towards the nodes 9, 8 and 6 for a corner geometry and nodes 9, 6 and 3 for a side geometry. Any elements with other orientations must be geometrically transformed to achieve this. There are four types of transformations, defined as below.

1. *Horizontal mirroring* - a mirroring along a horizontal axis along nodes 8, 5 and 2.
2. *Vertical mirroring* - a mirroring along a vertical axis along nodes 4, 5 and 5.
3. *Descending diagonal (D-diagonal) mirroring* - a mirroring along a descending diagonal axis along nodes 7, 5 and 3.
4. *Ascending diagonal (A-diagonal) mirroring* - a mirroring along an ascending diagonal axis along nodes 9, 5 and 1.

The transformations have to be carried out in a maximum of two steps. The first step aligns the geometry at the correct corner or side, whilst the second step is carried out to correct the orientation of any asymmetries.

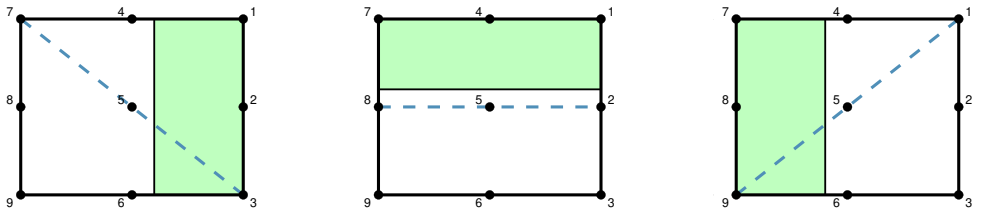
Starting with the corner geometry, it is clear that the initial step requires either a horizontal, vertical or descending diagonal mirroring. This is illustrated in fig. 3.9.



(a) Vertical mirroring. (b) D-diagonal mirroring. (c) Horizontal mirroring.

Figure 3.9: Corner geometry at different positions requiring different transformations to achieve desired level set orientation.

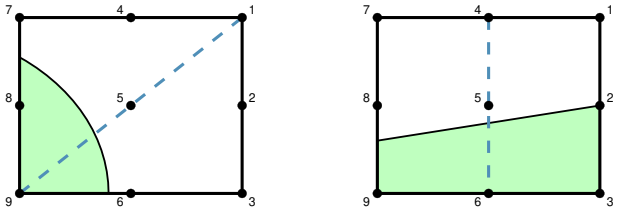
For the side geometries, the corresponding mirroring relevant for the first step is shown in fig. 3.10. These include the D-diagonal, horizontal and A-diagonal mirroring.



(a) D-diagonal mirroring. (b) Horizontal mirroring. (c) A-diagonal mirroring.

Figure 3.10: Side geometry at different positions requiring different transformations to achieve desired level set orientation.

The examples above are all symmetric, but this is seldom the case. Therefore, after the first step has been carried out, the second step is to determine where the weight is located in an asymmetric geometry. This may result in an extra transformation; an A-diagonal for the corner geometry or a vertical for a side geometry. This is illustrated in fig. 3.11.



(a) A-diagonal mirroring for corner geometry. (b) Vertical mirroring for side geometry.

Figure 3.11: Asymmetric corner and side geometry requiring different transformations to achieve desired level set orientation.

To achieve the desired level set value orientation in an element, the level set vector has to be permutated in accordance with above defined transformations. That means that any level set values that are interchanged in the mirroring has to be interchanged in the vector. As an example, performing a vertical mirroring permutates the level-set vector in eq. 3.6 to

$$LS^k_{vertical} = \{ls^k_7, ls^k_8, ls^k_9, ls^k_4, ls^k_5, ls^k_6, ls^k_1, ls^k_2, ls^k_3\} \tag{3.9}$$

since the level set values at node 1 has been interchanged with the level set value node 7, and so on.

The process of determining the orientation and applying the permutation must be performed for all relevant elements. This is done by evaluating all corner and side trios of nodal level set values and comparing their sums. That is, if $(ls^k_4 + ls^k_7 + ls^k_8) > (ls^k_9 + ls^k_6 + ls^k_3)$, then it is more likely to be an upper right corner geometry rather than a lower side geometry. This comparison is carried out for all trios, and the maximum sum determines type and position of geometry, and thus also which transformations are to be carried out.

3.3.4 Permutation of mass matrix

Each level set value in the level set vector is associated to a node and each local mass matrix entry is associated to two nodes. This means that any permutation of the level set values will result in a corresponding permutation of the mass matrix entries. Taking the vertical permutation in eq. 3.9 would correspond to the local mass matrix M^K on the element K as defined by

$$M^K_{vertical} = \begin{bmatrix} M^K_{77} & M^K_{78} & M^K_{79} & M^K_{74} & M^K_{75} & M^K_{76} & M^K_{71} & M^K_{72} & M^K_{73} \\ & M^K_{88} & M^K_{89} & M^K_{84} & M^K_{85} & M^K_{86} & M^K_{81} & M^K_{82} & M^K_{83} \\ & & M^K_{99} & M^K_{94} & M^K_{95} & M^K_{96} & M^K_{91} & M^K_{92} & M^K_{93} \\ & & & M^K_{44} & M^K_{45} & M^K_{46} & M^K_{41} & M^K_{42} & M^K_{43} \\ & & & & M^K_{55} & M^K_{56} & M^K_{51} & M^K_{52} & M^K_{53} \\ & & & & & M^K_{66} & M^K_{61} & M^K_{62} & M^K_{63} \\ & & & & & & M^K_{11} & M^K_{12} & M^K_{13} \\ & & & & & & & M^K_{22} & M^K_{23} \\ & & & & & & & & M^K_{33} \end{bmatrix} \tag{3.10}$$

with $M^K_{vertical}$ symmetric such that $M^K_{ij} = M^K_{ji}$. Therefore, the permutation of the level set values requires an inverse permutation of the local mass matrix. This is performed using the index vector I_{sort} as given by

$$LS^K = LS^K_{vertical}(I_{sort}) \tag{3.11}$$

such that

$$M^K = M^K_{vertical}(I_{sort}, I_{sort}) \tag{3.12}$$

which allows the local matrix to be assembled into the global mass matrix and a corresponding stiffness matrix.

3.4 Architecture of the networks

The procedure of generating an optimal multi layered perceptron (MLP) combines some general benchmarks with the fine method of *trial and error*. A variety of networks must thus be properly tested before reducing the set to a final and functioning choice. Two networks are relevant for this study, one for classification (Cnet) and one for regression (Rnet). Both are feedforward networks with back propagation algorithms to adjust the weights and biases according to feedback.

There are a few key characteristics associated in the construction of a classification network. First and foremost, hidden layers in the network are all equipped with an activation function which transforms the the level of neural activation to a numeric response. As a rule of thumb, a smooth, differentiable function within the range $[0, 1]$ is beneficial to use for classification as it can be interpreted as a probabilistic measure. As found in, for instance, [18] a sigmoid function

is a suitable choice for a classification data set. The function is defined by

$$f_{sig}(x) = \frac{1}{1+e^{-x}}. \tag{3.13}$$

The number and sizes of hidden layers must also be determined. This can be related to the sizes of the input and output layers, which are directly dependent on the size of the data. As has previously been discussed, the task of Cnet is to determine whether the element is *cut*, *exterior* or *interior* given its 9 level set values. MATLABs *patternnet* function requires the output to be numerical, hence the three classes can be implemented as a discrete and given by $\{1,2,3\}$, or binary and defined by $[1,0,0]$, $[0,1,0]$, $[0,0,1]$. For a probabilistic interpretation of the values, the binary classification can be used, resulting in an input size of 9 and an output size of 3. The hidden layers can thus be chosen to be of decreasing size.

A common choice of performance function is the mean squared error (MSE) function, which is given by

$$MSE = W\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{3.14}$$

where n is the number of observations, $x_i \in \mathbb{R}^m$ is the observed value, $\hat{x}_i \in \mathbb{R}^m$ is the approximated value and W denote error weights (which are to be discussed later). The observations can thus consist of a set of m labels per element. The MSE function is to be minimised using some optimisation algorithm, with the Levenberg-Marquardt (LM) algorithm as presented in [19, 20] being common. It is a trust region type algorithm best compared to a combination of a steepest descent with a Gauss-Newton method. The derivation stems from a first order Taylor expansion of a minimisation function $F(x)$ providing the Newton function, which for the Levenberg-Marquardt algorithm is modified to a step direction function which updates the current guess according to the recursive function given by

$$x_{k+1} = x_k - (J^T(x_k)J(x_k) + \lambda_k I)^{-1}J^T(x_k)r(x_k) \tag{3.15}$$

with $J(x)$ being the Jacobian of the minimisation function, $r(x)$ is the residual function, I the identity matrix and x_k is the guess of the current iteration. λ_k is a damping parameter related to the steepest descent, $J^T(x)r(x) \approx \nabla F(x)$ is an approximation of the gradient of the minimisation function $F(X)$, whilst $J^T(x)J(x) \approx \nabla^2 F(x)$ is an approximation of the corresponding Hessian. The weights and biases of the network are updated according to this scheme for each training iteration.

There are less benchmarks for the construction of a regression network. In regards of activation functions, it is common to use a rectified linear unit (ReLU) function due to its strength in the case of a vanishing gradient during network training. It is given by

$$f_{ReLU}(x) = \max(0, x) \tag{3.16}$$

However, a smooth function may be useful when the data at hand is of varying magnitude as in this study, which re-introduces the sigmoid function as well as a hyperbolic tangent function

and a softmax function. The hyperbolic tangent is given by

$$f_{tan}(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{3.17}$$

and the softmax by

$$f_{soft}(x) = \frac{e^x}{\sum_{i=1}^{n} e^{x_i}} \tag{3.18}$$

for $x \in \mathbb{R}^n$ which can only be applied on discrete input. Some general notes regarding some common activation functions are outlined in [21, 22].

The architecture of the hidden layers can be constructed based upon a similar argument as the classification network. The input size is likewise 9, however, the output consists of the entries of the local mass matrix. The local mass matrix consists of $9^2 = 81$ entries for second order elements, however, since the matrix is symmetric this reduces to $(81-9)/2+9 = 45$ elements. Hence, the output size of the regression network ought to be 45 and the hidden layers can thus constitute an increase from 9 to 45. Given that the difference in input and output size is relatively large, it may indicate that at least two, albeit likely three, layers are required.

The LM method combined with the MSE function can be used for the regression network as well, for it is generally a suitable choice regardless of network type.

It may be necessary to introduce *error weights* into the training when the output varies remarkably in magnitude, as in this study. The error weights determine the importance of the error which is generated at a given data point, and allow for a focused training on certain data points as specified by the user. There are two ways (three if combined) the weights can be applied (other than $W = 1$ if they are not used). One way is to weigh the different outputs in a sample differently, such that certain outputs are especially important during training. In this regression study, this would correspond to weighing the 45 local mass matrix entries differently based upon which entries that require extra training. One other way is to weigh each sample differently, which in this study would correspond to weighing all 45 entries per element equally, although weighing different sets of 45 elements within the training data differently. That is, for a set of data points X with $\dim(X) = [m, n]$, the error weights W can be either $W \in \mathbb{R}^m$, $W \in \mathbb{R}^n$ or $W \in \mathbb{R}^{m \times n}$.

CHAPTER 4

METHOD AND IMPLEMENTATION

The previous chapters have outlined information regarding CutFEM, some mathematical finesse and neural networks which are all required for the implementation as well as the decisions and motivations that follow with it. The ultimate goal is to produce a method as described in algorithm 2, although this work requires the networks to be implemented and tested individually for a sufficient evaluation of the individual networks.

The implementations have been entirely carried out in MATLAB. There have been an emphasis on the Statistics and Machine Learning toolbox as this includes the functions for creating and using neural networks.

4.1 Producing and filtering the data set

The full, unfiltered data set was generated for the three geometries as discussed in section 3.2. A mesh size of $h = 0.05$ was used for all geometries, for whilst this is eventually scaled to a reference element it does have an impact on the number of elements as generated for each geometry.

The radius increment and the center coordinates can be changed to achieve different domains of the circular geometries. The radius increments were set to

$$dR_r(h) = \frac{h}{10}, \quad dR_b(h) = \frac{h}{5}, \quad dR_g(h) = \frac{h}{2} \tag{4.1}$$

in this study, which satisfies eq. 3.4 and the subscripts denote the same zones as shown in fig. 3.2. The same was applied to both circles and holes. The radius had an allowed range defined by $[dR_r(h), \sqrt{0.5} - dR_g(h)]$ as any larger radius would completely fill the background mesh and thus generate no intersections.

Five random center coordinates were produced for each iteration of the radius increment to achieve the variation of center coordinates as described in eq. 3.3. This means that five different circles and holes were produced for each radius as included in the study. For the planar set, the

iteration instead occurs over the coefficient a in a step $da(h)$ and the level constant d in step $dd(h)$. These were defined to

$$da(h) = dd(h) = \frac{h}{10} \tag{4.2}$$

for both.

Nine level set values was generated for each element on each specific domain. These were used as features in two data sets; one for classification and one for regression.

As discussed in section 2.2.1, there are no negative contribution from the spline functions, ensuring that any element with all positive nodes is on the interior of the domain, and vice versa. These elements are thus not of relevance for the classification problem, since these can be classified without passing them through a classification network. Thus, these known elements were excluded from the data set to reduce the dimensionality of the classification problem, whilst the rest of the elements were classified to generate the corresponding elemental labels. The labels were set to binary classification labels, as discussed in 3.4, and a corresponding key was generated to keep track of the categorical representation. The resulting data set constitutes the unfiltered classification data set.

For the regression data set, the 45 unique mass matrix entries were sought as labels to the data set. Only the elements that actually cut the boundary of the domain, that is $T \in \mathcal{T}_{h,C}$, was included to reduce the task of the regression network. This is based upon the assumption that Cnet will work sufficiently well and only the intersecting elements will pass on to Rnet. The elements on the exterior $T \in \mathcal{T}_{h,E}$ need not to be approximated since these have zero contribution, whilst the elements on the interior $T \in \mathcal{T}_{h,I}$ have the same contribution and can thus be pre-computed and applied for all.

The features of both networks as well as the labels of Rnet were filtered by the processes as described in section 3.3. Although the inverse of the scaling and permutation of the mass matrix elements had to be applied for the data set to generate the desired form of the labels.

Lastly, the filtered data was to be partitioned into a training and a testing set. This was performed for both the Cnet data set and the Rnet data set, with 85% to the training set and 15% to the testing set. The partition was performed randomly and a few different partitions were made for the regression set to later investigate the effect. The quality of the results may differ for different partitions since they were performed randomly and it is therefore unknown on beforehand whether a good or a bad selection would be generated. It would be beneficial to include elements with especially large variance in the training set, and some random partition may or may not sufficiently represent this. Further, some data sets were created in which any duplicates in the feature set up to a determined tolerance were removed.

4.2 Constructing the networks

A few areas were investigated in the construction of the neural networks. These are shortly outlined below for the most important steps.

4.2.1 Optimisation of Cnet

A variety of 1- and 2-layer classification networks were created and trained. This included some of decreasing hidden layer size, as previously mentioned, but also a few other variations for testing purposes. The results did not differ tremendously between the different architectures, but eventually two hidden layers of sizes 7 and 5 were chosen. The sigmoid activation function as discussed was used for all classification networks. The final choice of Cnet is illustrated in fig. 4.1.

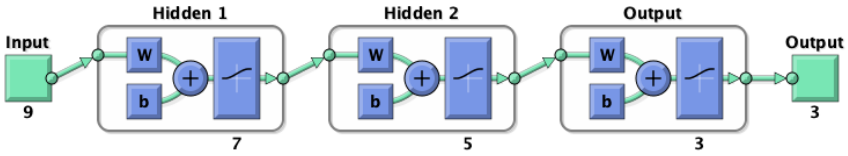


Figure 4.1: Final architecture of Cnet.

4.2.2 Optimisation of Rnet

The construction of Rnet was a far more extensive task than that of Cnet. This was performed in multiple steps to evaluate one property at the time.

First, a suitable activation function was to be chosen. All activation functions as outlined in section 3.4 were used to train a one layered network of hidden layer size 10. The training did only converge for the hyperbolic tangent function, since all algorithms based on other functions aborted due to minimum gradient reached. Some variations on the architecture was performed to study whether the error stemmed primarily from the network only consisting of one hidden layer, although since the results were not sufficiently improved the hyperbolic tangent function was eventually settled upon and used.

Moving forward, the number of hidden layers and their sizes were investigated. This was performed primarily for networks consisting of two or three hidden layers, however, a few networks of a singular hidden layer were also included. The lowest testing MSE was eventually found to occur for a network of three hidden layers of sizes 15, 25 and 35.

As this was further studied, there appeared to be an issue with the magnitudes of the mass matrix approximations. Several networks of the same architecture were created to investigate any issues in stemming from any subsets of the 45 labels per element. Each of these networks were trained on the different subsets, and the results were evaluated to identify any patterns in the difficulties. Below are a few examples of what labels of subsets were tested.

1. One label corresponding to some specific matrix entry at M^K_{ij}, where the indices $i, j \in \{1,2,..,9\}$ were tested for a few different alternatives.

2. One label corresponding to the elemental area as computed from the local mass matrix.

3. Nine labels corresponding to the diagonal of the local mass matrix M^K_{ii} for $i = 1,2,..,9$.

And further, some element-wise tests were performed to investigate how the regression performed for different elemental cuts.

The above investigation showed that the middle to lower valued contributions in the local mass matrices were badly approximated, sometimes to several orders of magnitude. This passes the training and testing since the generally low values passes of as a low MSE, despite being orders of magnitude wrong. This may severely affect the final method, especially if the values are highly over approximated. Because of this, it was of uttermost importance to induce a focus towards the MSE of these lower values during training, so that they were not overlooked despite their low error contribution. This was performed by introducing error weights to the training data.

The idea with the error weights was to upscale the impact of the low valued entries. These entries were mostly element related, meaning that an element with generally low mass matrix values was the most prone to errors. This introduced the idea that the error may be related to any cuts between the error and domain with lower intersection area. Thus, the error weights could be chosen element-wise as inversely proportional to the elemental area, given by

$$W_k = (a^K)^{-1} \tag{4.3}$$

where a^K denotes the elemental area of element K which can be computed using quadrature or according to the method which will be presented in section 4.3.3.

The results improved with the implementation of the error weights and these were therefore used. Some comparison tests were performed to study the effect of the weights as well the effect of the filtering of the data, both which proved to be crucial in the training as proven in the convergence of the training.

The final choice of Rnet is illustrated in fig. 4.2.

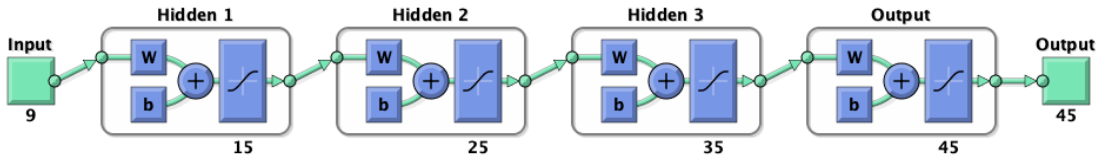


Figure 4.2: Final architecture of Rnet.

4.3 Performance test

The performance tests were carried out differently for the different networks depending on which properties that proved to be of relevance to investigate further. A few of the performance measurements which were carried out are as defined below. The model problem in eq. 2.1 -2.2 was used with the parameters

$$c = 1, \quad f(x,y) = 2\sin(x)\sin(y) \tag{4.4}$$

and the Neumann boundary condition

$$g_N = \mathbf{n} \cdot (\cos(x)\sin(y), \sin(x)\cos(y)) \tag{4.5}$$

which corresponds to the solution

$$u(x, y) = \sin(x)\sin(y). \tag{4.6}$$

4.3.1 Algorithmic implementation

The algorithm as shown in 2 is the ultimate goal of the method, however, two different algorithms were created as the networks ought to be tested separately for best evaluation. For Cnet this is outlined in algorithm 3.

Algorithm 3 CutFEM using Cnet.

$\mathcal{T}_{h,0} \leftarrow$ background mesh
$ls \leftarrow$element-wise level set values
$F \leftarrow$ filtering functions
$\tilde{ls} \leftarrow F(ls)$
$\{\mathcal{T}_{h,E}, \mathcal{T}_{h,C}, \mathcal{T}_{h,I}\} \leftarrow$ Cnet$(\tilde{ls})$
$M^K(\mathcal{T}_{h,C}) \leftarrow$ quadrature
$M^K(\mathcal{T}_{h,I}) \leftarrow$ pre-computed
$M \leftarrow$ assemble M^K

The above algorithm shows that quadrature only has to be applied on the elements that are determined to actually cut the boundary. The corresponding algorithm for Rnet is displayed in 4 and quadrature is used to determine which elements are cut, yet removed from a computational loop over the cut elements.

Algorithm 4 CutFEM using Rnet.

$\mathcal{T}_{h,0} \leftarrow$ background mesh
$ls \leftarrow$element-wise level set values
$F \leftarrow$ filtering functions
$\tilde{ls} \leftarrow F(ls)$
$\{\mathcal{T}_{h,E}, \mathcal{T}_{h,C}, \mathcal{T}_{h,I}\} \leftarrow$ quadrature
$M^{\hat{K}} \leftarrow$ Rnet$(\tilde{ls}(\mathcal{T}_{h,C}))$
$M^K(\mathcal{T}_{h,C}) \leftarrow F^{-1}(M^{\hat{K}})$
$M^K(\mathcal{T}_{h,I}) \leftarrow$ pre-computed
$M \leftarrow$ assemble M^K

4.3.2 Benchmark domains

Both of the networks were tested on three benchmark domains. The idea with the benchmark domains was to investigate how any error propagated for different curvature on the domain.

A circular domain was of relevance to be included since the data set was partially based upon such. The level set of the circle is as shown in eq. 3.1. The parameters were set to $x_c = y_c = 0.5$ and $R = 0.3$ throughout the testing.

An elliptical domain was also used to include some slight variation on the curvature of the circular domain. The level set is given by

$$\phi_e(x,y) = R^2 - \frac{(x-x_c)^2}{2} - 2(y-y_c)^2 \tag{4.7}$$

with the same parameters as defined for the circle.

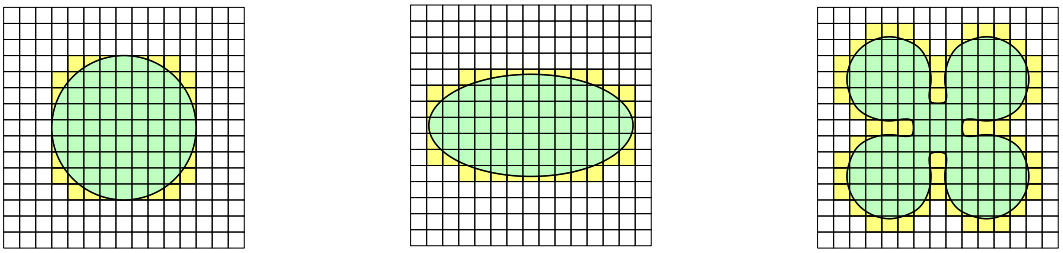
A *flower* geometry was also included to generate a domain with heavy curvature and boundary sections approaching an edge yet still maintaining the rounded shape. The level set of the flower is essentially consisting of the maximum of five circles as given by

$$\phi_f(x,y) = \max(\phi_f^{(1)}, \phi_f^{(2)}, \phi_f^{(3)}, \phi_f^{(4)}, \phi_f^{(5)}) - 0.0085, \tag{4.8}$$

$$\begin{aligned}
\phi_f^{(1)}(x,y) &= \frac{R^2}{2} - (x - \frac{c_1+c_2}{2} + \xi_x)^2 - (y - \frac{c_1+c_2}{2} + \xi_y)^2, \\
\phi_f^{(2)}(x,y) &= R^2 - (x - c_1 + \xi_x)^2 - (y - c_1 + \xi_y)^2, \\
\phi_f^{(3)}(x,y) &= R^2 - (x - c_2 + \xi_x)^2 - (y - c_1 + \xi_y)^2, \\
\phi_f^{(4)}(x,y) &= R^2 - (x - c_1 + \xi_x)^2 - (y - c_2 + \xi_y)^2, \\
\phi_f^{(5)}(x,y) &= R^2 - (x - c_2 + \xi_x)^2 - (y - c_2 + \xi_y)^2
\end{aligned} \tag{4.9}$$

where c_1, c_2 denote some center coordinates of the circles, and $\xi_x, \xi_y \in h[-0.5, 0.5]$ are evenly distributed stochastic variables with h denoting the mesh size. The stochastic variables allow for a perturbation of the center coordinates, which may generate a variation of cuts. These can be manually set to zero for a centered domain. $c_1 = 0.3$, $c_2 = 0.7$ and $R = 0.2$ were used for the flower.

The three domains are illustrated in fig. 4.3 with an example mesh size of $h = 0.1$. Any sharp edged geometries were excluded since this was not included in the training data.



(a) Circle. (b) Ellipse. (c) Flower.

Figure 4.3: Three benchmark domains used for performance tests.

4.3.3 Measurement of errors

A few measurements of errors are of interest for the two networks when comparing them to an analytical solution as well as a CutFEM-based method. Below are a few of these defined, other than the error estimates and expected bounds as defined in section 2.3.3.

As have been discussed in section 2.2.1, any function in the space can be expressed as a linear combination of the basis functions. This is up to second order polynomials for the second order case. There are some information to be extracted from lower order linear combinations too, which can be used for testing the method in steps of increasing order. One measure of interest is the area, which can be derived from a zeroth other linear combination. The function expansion in the zeroth order yields zeroth order coefficients of ones which can be used to compute the area using

$$a^K = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \cdot M^K \cdot \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{4.10}$$

where M^K denotes the local mass matrix. A global measure can be performed to obtain the domain area a by using the global mass matrix M instead and a reference measure can be performed using the reference mass matrix $M^{\hat{K}}$. The relative error is then given by

$$E^{K,rel}_{area} = \frac{|a^K - a^K_{ana}|}{a^K_{ana}} \tag{4.11}$$

where a^K_{ana} is the analytical area.

Based upon a first order linear combination, the center of mass (CM) can be derived using certain coefficients. These coefficients are related to the spline element nodal coordinates. The computation can be performed for the specific element M^K, however, for a relevant comparison this is better carried out on the the reference element $M^{\hat{K}}$ and with corresponding reference spline element coordinates as coefficients. The CM coordinates are thus computed from

$$CM^{\hat{K}} = \frac{1}{a^{\hat{K}}} \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \cdot M^{\hat{K}} \cdot \begin{bmatrix} 1.5 & 1.5 & 1.5 & 0.5 & 0.5 & 0.5 & -0.5 & -0.5 & -0.5 \\ 1.5 & 0.5 & -0.5 & 1.5 & 0.5 & -0.5 & 1.5 & 0.5 & -0.5 \end{bmatrix}^T \tag{4.12}$$

and the results of the Rnet approximation can be compared to the corresponding results using quadrature.

One last general error measure which was studied constituted of a relative local mass matrix entry error. This was computed from

$$E^{K,rel}_{ij} = \frac{|M^{K,RN}_{ij} - M^{K,CF}_{ij}|}{M^{K,CF}_{ij}} \tag{4.13}$$

where $M^{K,RN}$ denotes the local mass matrix as approximated by Rnet and $M^{K,CF}$ is the local mass matrix as computed using CutFEM.

CHAPTER 5

RESULTS

The initial tests of the networks consists of some general performance tests. With the results as a baseline, different characteristics have been investigated further to examine any reoccurring difficulties. Further, the different networks have been tested individually and with different techniques due to their different characteristics and depending on which properties which have revealed themselves during the phase. The networks are tested individually since it would not be straightforward to analyse any errors occurring from a combination of the networks (as mentioned in chapter 4). The results are accompanied by some brief comments on what can be observed and why any proceeding tests were performed, however, a further discussion of the results is carried out in chapter 6.

5.1 Classification results

For Cnet it is not obvious how to obtain a meaningful error comparison. This is because of below reasons.

1. It is not apparent which norm would provide a meaningful error estimation.

2. The dimensions of the resulting linear system may be different for the Cnet method, for the number of active elements may differ due to wrongful classifications.

For this reason, other performance comparisons are illustrated below which may provide some clarity on the overall success rate.

The relative error area of the entire domain was computed to evaluate the final mass matrix. This was done using the expression in eq. 4.11 but with the global mass matrix. The analytical area was set to the corresponding area of CutFEM using the same mesh size. The results are illustrated in tab. 5.1. It shows that the algorithm successfully classifies all elements for the circle and the ellipse, although it has some difficulties in classifying the elements of the flower. This

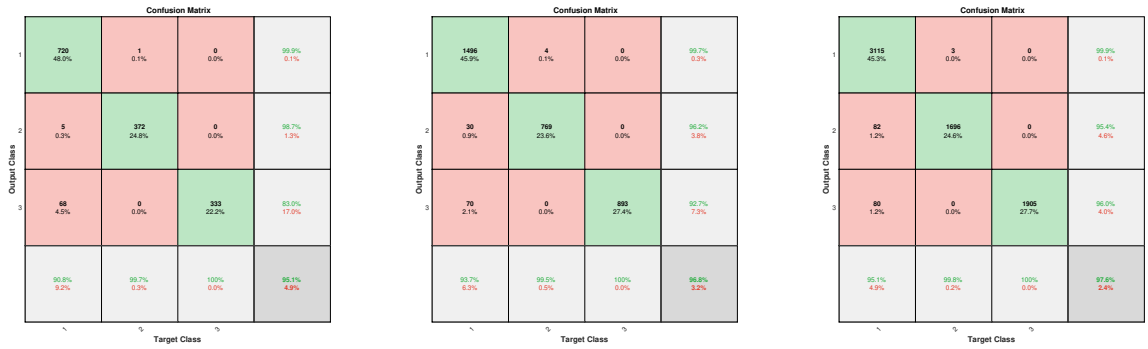
error decreases with mesh size, yet it is not apparent whether this is due to a better performance or a smaller contribution to to the areal error since the elements also become smaller with mesh size.

Table 5.1: Relative error of area computed from mass matrix using Cnet and CutFEM.

Mesh size	0.1	0.05	0.025
E^{rel}_{circle}	0	0	0
$E^{rel}_{ellipse}$	0	0	0
E^{rel}_{flower}	$4.08 \cdot 10^{-2}$	$1.14 \cdot 10^{-2}$	$2.98 \cdot 10^{-5}$

The success rate of the flower domain was further evaluated to investigate whether the results improved with a smaller mesh size. The circle and ellipse were excluded since it was apparent from the area study that these had a 100% success rate.

The flower was investigated by randomising the center coordinates ten times for each mesh size. This was performed to average any outliers occurring from certain variations on the domain. The results are displayed in confusion plots in fig. 5.1. It is apparent that there is an increase in success rate with a decreasing mesh size, indicating that the effect of the issue can be reduced.



(a) $h = 0.1$. (b) $h = 0.05$. (c) $h = 0.025$.

Figure 5.1: Success rate of Cnet on 10 flower geometries with randomised center coordinates for different mesh sizes.

The wrongfully classified elements were investigated for a mesh size of $h = 0.1$ to determine where any errors may occur. The elements which were misclassified are tinted in red in fig. 5.2. It shows that the innermost elements, where the circles meet, appear to be difficult to classify. The same trend is apparent for the smaller mesh sizes. The Cnet classification for some mesh sizes and the corresponding quadrature classifications are displayed in appendix A

5.2 Regression results

The results for Rnet consist of two parts. The first concludes some final investigations which are relevant for final motivations and any further suggestions for updates of this work. The second

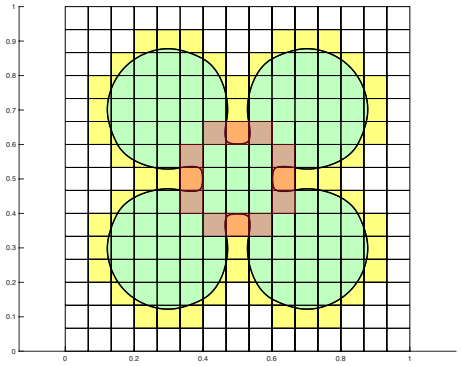


Figure 5.2: Flower geometry with mesh size $h = 0.1$. The wrongfully classified elements by Cnet are marked in red tint. Green denotes the domain and the yellow elements are cut elements.

is a concluding error analysis by studying the model problem. The errors from this are then investigated in a final data science analysis.

5.2.1 Effect of variations

The results of the different data partitions were investigated to examine if the extracted elements had any effect on the final tests. The error in L^2-norm was computed for the model problem and the results are displayed in tab. 5.2. The partition variation *factor* denotes a network which was trained upon a data set multiplied with a factor of $1/1.05125$ and was originally included by mistake. The output is multiplied by the reciprocal of the factor to obtain the correct values.

Table 5.2: Error in L^2-norm for variations of regression networks.

Partition variation	Error
None	$1.89 \cdot 10^{-2}$
Factor	$2.86 \cdot 10^{-5}$
Tolerance 1	$9.59 \cdot 10^{-3}$
Tolerance 2	$1.72 \cdot 10^{12}$

The table shows that the best results occur for the data set multiplied by the factor. The results overall display some large variations, although one of the networks trained on the data with a tolerance based duplicate removal appears to be have an exploding error. This may be dependent directly on the partition, or by some unforeseen optimisation process which have gone wrong during training. Based upon the above results, the network trained using the factored data set was used for all computations.

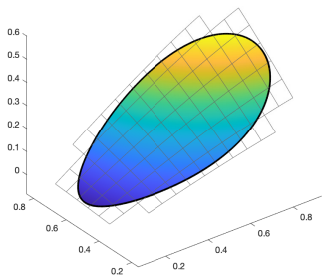
The load vector cannot be computed conventionally if the algorithm is to be completely free of unnecessary quadrature. This can be overcome by making a L^2-projection of the load function onto the mesh, and multiplying it with the mass matrix specific to the domain, that is, the approximated mass matrix. This was briefly mentioned in section 2.3.1. Whilst the goal is to use this method, a comparison between the results of this quadrature free method and a

conventional method can be compared. This was performed for the model problem as well as a L^2-projection of the solution and the resulting error in L^2-norm is displayed in tab. 5.3. The domain is the elliptic one since this showed some error prone results, and the mesh size was set to $h = 0.1$.

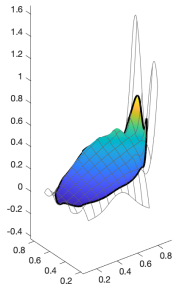
Table 5.3: Error of L^2-projection and model problem for different methods of computing the load vector. A CutFEM benchmark is included.

Origin of load vector	L^2-projection	Model problem error
Mass matrix	$6.96 \cdot 10^{-7}$	$1.44 \cdot 10^{-3}$
Quadrature	$1.05 \cdot 10^{-2}$	$8.94 \cdot 10^{-4}$
CutFEM benchmark	$7.27 \cdot 10^{-7}$	$5.11 \cdot 10^{-7}$

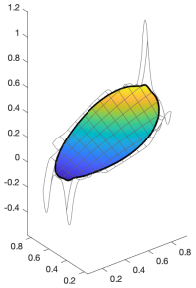
From the results it is not apparent which method produces the best results, but the error for the quadrature based load vector is remarkably large in the L^2-projection. The corresponding graphical results are illustrated in fig. 5.3. The L^2-projection with quadrature assembly displays some major difficulties, whilst the model problem does not seem to differ much between the methods. The analytical solution is displayed in fig. A.3.



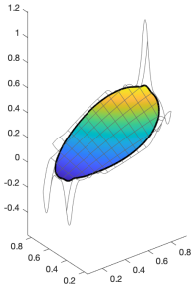
(a) L^2-projection, mass matrix.



(b) L^2-projection, quadrature.



(c) Model problem, mass matrix.

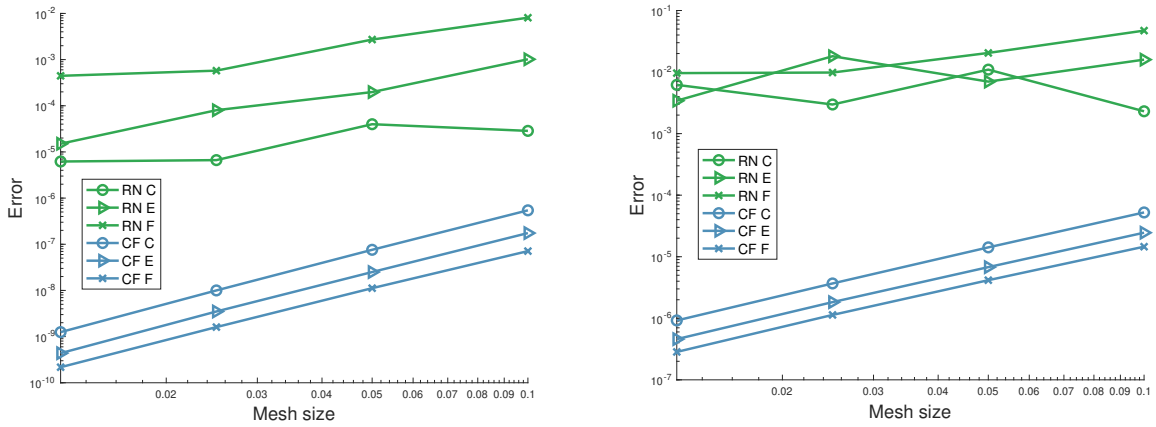


(d) Model problem, quadrature.

Figure 5.3: Graphic illustration of different methods of load vector assembly for L^2-projection and model problem on ellipse of mesh size $h = 0.1$.

5.2.2 Error convergence and analysis

The error study was performed on the model problem for all geometries and the mesh sizes $h = 0.1, 0.05, 0.025, 0.0125$. The error was computed in L^2-norm and energy sub-norm for both Rnet and CutFEM. The results are illustrated in fig. 5.4. We know from section 2.3.3 that the error in L^2-norm converges to rate of third order and the error in energy sub-norm converges to a rate of second order for CutFEM. We do note some convergence in L^2-error with mesh size for Rnet, however not as fast as for the CutFEM algorithm. The error in energy sub-norm is of a more static character.



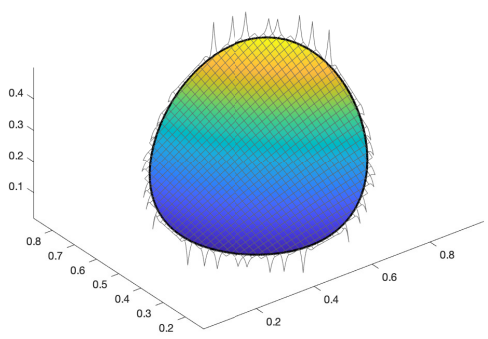
(a) L^2-norm error.

(b) Energy sub-norm error.

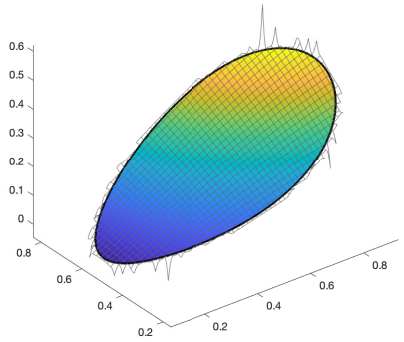
Figure 5.4: Error in L^2-norm and energy sub-norm for the regression network (RN) and CutFEM algorithm (CF) over mesh size and geometries circle (C), ellipse (E) and flower (F).

The solutions on all three geometries were graphically illustrated to investigate any error-prone areas. This was performed for a mesh size of $h = 0.025$. The results are displayed in fig. 5.5. The overall solutions appear to be approximated decently, however, there are some instabilities along the boundary.

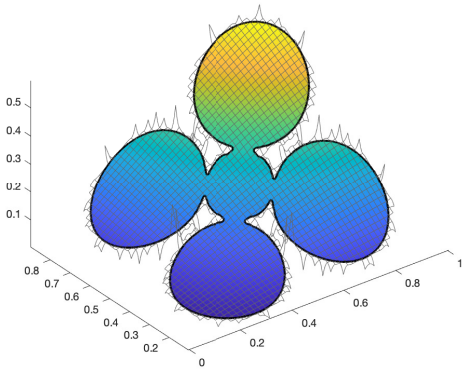
A node-wise comparison was made between Rnet and CutFEM to investigate the origin of any errors. This was performed for the ellipse of mesh size $h = 0.05$, which had shown some instabilities despite its rounded boundaries. This was done by comparing the 45 estimated matrix entries to the 45 matrix entries as computed by CutFEM. The 45 entries were stored in a



(a) Circle.



(b) Ellipse.



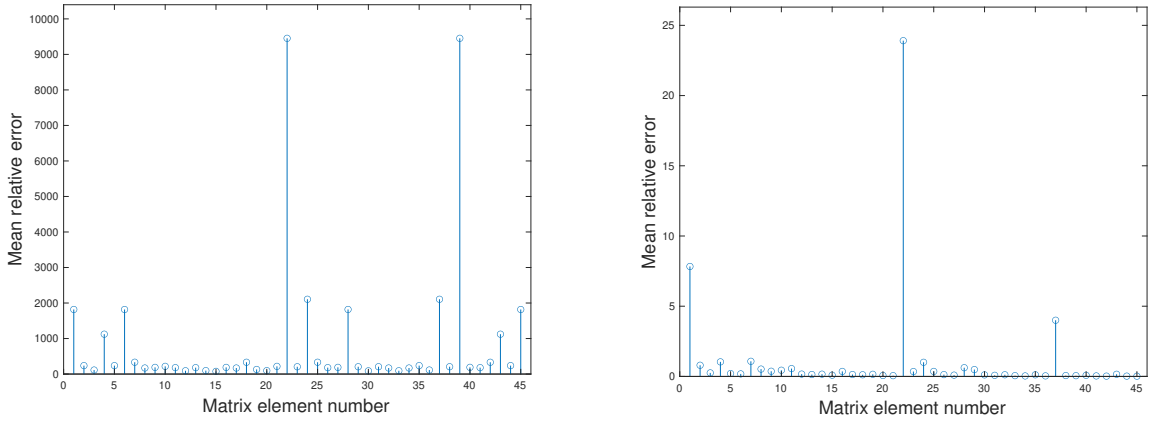
(c) Flower.

Figure 5.5: Graphical illustration of solution for Rnet of three geometries and mesh size $h = 0.025$.

vector $m_K \in \mathbb{R}^{45}$ and the corresponding position in the local mass matrix M^K is illustrated in

$$M^K = \begin{bmatrix} m_K(1) & m_K(2) & m_K(4) & m_K(7) & m_K(11) & m_K(16) & m_K(22) & m_K(29) & m_K(37) \\ & m_K(3) & m_K(5) & m_K(8) & m_K(12) & m_K(17) & m_K(23) & m_K(30) & m_K(38) \\ & & m_K(6) & m_K(9) & m_K(13) & m_K(18) & m_K(24) & m_K(31) & m_K(39) \\ & & & m_K(10) & m_K(14) & m_K(19) & m_K(25) & m_K(32) & m_K(40) \\ & & & & m_K(15) & m_K(20) & m_K(26) & m_K(33) & m_K(41) \\ & & & & & m_K(21) & m_K(27) & m_K(34) & m_K(42) \\ & & & & & & m_K(28) & m_K(35) & m_K(43) \\ & & & & & & & m_K(36) & m_K(44) \\ & & & & & & & & m_K(45) \end{bmatrix}. \tag{5.1}$$

The relative nodal error was computed as in eq. 4.13 for each element, and the mean was taken over all samples. This was performed for a mass matrix without permutation as well as a mass matrix which was permutated such that the heaviest level set values were distributed towards the lower, right side. See fig. 3.8 for nodal numbers. The results are displayed in fig. 5.6.



(a) Not permutated.

(b) Permutated.

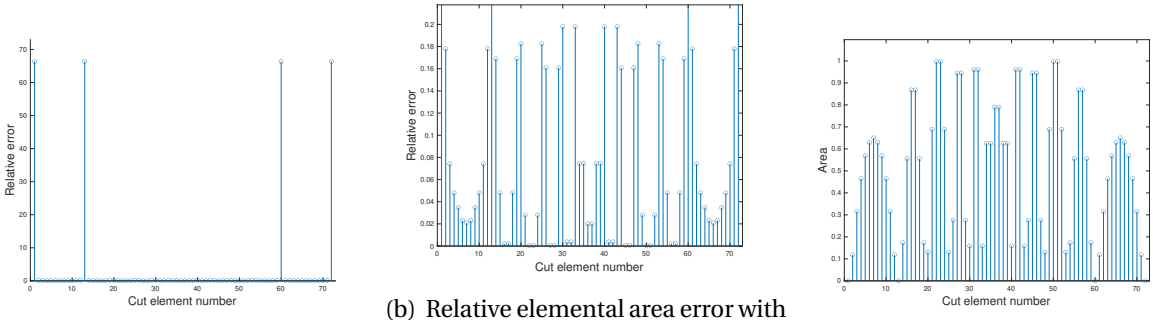
Figure 5.6: Mean nodal relative error for ellipse and mesh size $h = 0.05$.

For the case without permutation, the values with remarkably higher mean relative error are related to $m_K(22) = M_{17}^K$ and $m_K(39) = M_{39}^K$ which are related to the corners of the element. Further, $m_K(1) = M_{11}^K$, $m_K(6) = M_{33}^K$, $m_K(24) = M_{37}^K$, $m_K(28) = M_{77}^K$, $m_K(37) = M_{19}^K$, $m_K(45) = M_{99}^K$ are also higher than most of the values, and also related to the corners of the element.

When the values have been permutated, the domain intersection of the element would generally be located towards the nodes $9,6,8,3$, and the domain exterior to nodes $1,4,2,7$. From the figure we see that $m_K(1) = M_{11}^K$, $m_K(22) = M_{17}^K$ and $m_K(37) = M_{19}^K$ attain high values. These values are all related to node 1, which is the node supposed to be located the very furthest from the intersection to the domain after permutation. This indicates that the nodes exterior of the domain appears to be the most difficult to approximate.

To investigate any elemental errors, the elemental area and center of mass were computed on the reference element $\hat{K}$. The sample domain and mesh size were used as previously. The relative error was computed using eq. 4.11 and the results are displayed in fig. 5.7. The relative error seems to be nearly the inverse of the area, showing that smaller area yields larger errors, and vice versa. There are four distinguished peaks in the plot, these peaks are related to the elements with peaking values in fig. 5.3. These elements have an intersection area close to zero, as demonstrated in the area plot.

The center of masses on the elements of same sample domain were computed to further investigate the elemental errors. This was computed using eq. 4.12. Since the elemental area was used, any errors stemming from the zeroth order study are carried onto the center of mass. The results of the CM using Rnet and quadrature as well as the error are displayed in fig. 5.8. CM coordinates near $(0.5,0.5)$ indicates nearly fully domain intersecting elements, whilst coordinates near $x, y \in \{0,1\}$ indicate minor intersection with the domain. The figure shows that the further from the center the CM is located, the greater is the displacement. This essentially confirms what has previously been noted; the elements with minor intersection are the most difficult to approximate. The CM error follows a similar shape as the the area



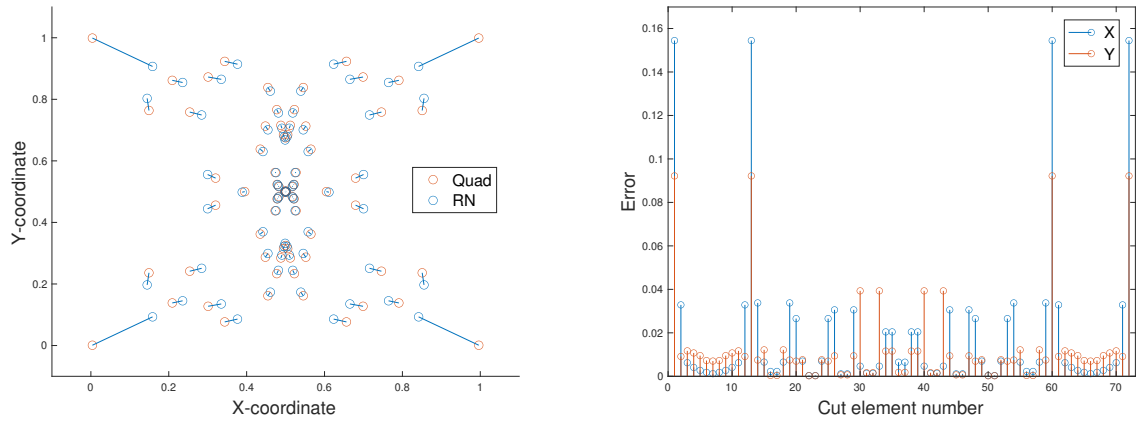
(a) Relative elemental area error.

(b) Relative elemental area error with adjusted axes.

(c) Elemental area.

Figure 5.7: Relative elemental area error and corresponding area for ellipse and mesh size $h = 0.05$.

error. The X-coordinates seem to contribute the most for large errors and vice versa. This may however only be a coincidence occurring from the specific domain.



(a) CM coordinates.

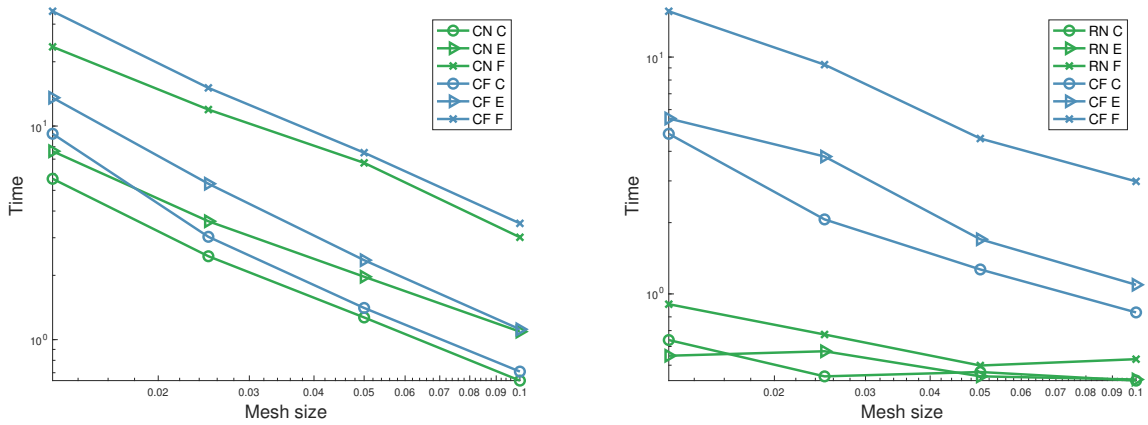
(b) CM error.

Figure 5.8: Center of mass (CM) computed using Rnet (RN) and quadrature (Quad) in coordinates and corresponding errors.

5.3 Time consumption

As discussed, the networks are tested individually and so are the measurements on the time consumption. The results for the networks in comparison to CutFEM are displayed in fig. 5.9.

Cnet shows slightly reduced computational time, which becomes more prominent for smaller mesh sizes (that is, a greater number of elements). The difference is increased further for Rnet, in which the network seems to barely be affected by an increased number of elements.



(a) Cnet.

(b) Rnet.

Figure 5.9: Time consumption of classification (Cnet, CN) and regression (Rnet, RN) network in comparison to CutFEM (CF) over different mesh sizes. The geometries used are circle (C), ellipse (E) and flower (F).

CHAPTER 6

DISCUSSION

The results which have been presented are discussed in this section. The disposition closely follows that of chapter 5 for easy reference. Some new ideas based upon the results that have been investigated will be presented and outlined for any future work on this study.

6.1 Evaluation of Cnet

The wrongfully classified elements by Cnet are displayed in tab. 5.1 and fig. 5.1. They may stem from two reasons.

1. The curvature of the innermost elements of the flower may not be largely represented in the training data.
2. The level set equation for the flower is a maximum of five level set equations (see eq. 4.8). This may generate discontinuities in the level set function on certain elements. These discontinuities likely result in odd combinations of level set values on these elements and it is likely that the network has not previously been trained upon them.

If the error would have been dependent on (1) alone, we would likely have noted a more prominent decrease in the error with the smaller mesh sizes and the problem would likely have vanished for the smallest mesh size since all elemental geometries are similar to circles or planes at that point (see fig. A.1 and fig. A.2). Therefore it is likely that the error is partially or entirely dependent on (2). This issue would therefore be reduced if multi-function level set representations would be included in the training set, or if these types of representations were entirely prohibited from being used with Cnet. Similar geometries may instead be interpreted using some other method, for instance a summation without taking the maximum of the values.

If the error still remains after the effect of (2) has been reduced further steps can be taken. The most obvious approach is to increase the fraction of intersections with high curvature in

the data set. This would result in more training of the element type and may reduce the error. Another approach of increasing the training on high-curvature cuts is to use the method with error weights to increase the relevance of these classifications.

One third, and more general, approach to both problems (1) and (2) is also related to error weights. Looking at target class 1 in fig. 5.1 and comparing to the other classes, it appears that the most common type of misclassification is to classify cut elements as exterior elements. The second most common error is to classify cut elements as interior elements. Both of these are fatal errors since the interior and exterior elements are either neglected or pre-computed (see algorithm 3). It would therefore be better if a fraction of the exterior and interior elements would be wrongfully classified as cut, since the quadrature would determine the true elemental contribution [1]. This could be accomplished by applying weights to the output. The weights can be applied element-wise, although this might yield unfavourable results since it essentially only would result in more focused training on the cut elements. Instead, the specific entries in the binary classification can be weighted differently. For instance, if $[1,0,0]$ denotes a cut element, the corresponding weight per sample could be

$$W = \begin{bmatrix} w_1 & w_2 & w_2 \end{bmatrix}, \quad w_1 > w_2 > 0 \tag{6.1}$$

and would thus be the same for all samples. This weight would be applied to each sample error and hence increase the importance of the cut classification, introducing a bias which likely identifies elements which are difficult to classify as cut.

6.2 Evaluation of Rnet

The evaluation of Rnet consists of some initial remarks on the results, followed by an analysis of the error.

6.2.1 Remarks on the regression

The different magnitudes of the error in tab. 5.2 shows the importance of a good training set. It is not guaranteed that a representative partition is made when the partition is performed randomly, and subsets of high variance may be excluded from the training set. Small, but important, variations between similar samples may also be excluded when a tolerance measure is applied and duplicates up to the tolerance is removed.

Generating multiple partitions and training different networks to find the most suitable alternative is a most tedious task, since the network training for this type of task may take many days to finish. Instead, as suggested by [12] yet not implemented, an adaptive partition algorithm can be used to ensure that the training data is of sufficient variance.

The especially high error for the L^2-projection when the load vector is computed from quadrature, as shown in tab. 5.3, is somewhat concerning. The problem involving only the

[1] Note that this is only a certainty for the algorithm without Rnet, since Rnet is not trained on approximating the values of exterior and interior elements.

mass matrix is slightly more sensitive to inconsistencies in the approximated result, since the smoothing effect of the differentiated basis functions as present in the stiffness matrix is not applied. This allows any element-wise error to directly carry on to the result on the corresponding nodes. Therefore it would be great to improve the approximation further, however, it is not entirely surprising that the error displays such instabilities as shown in fig. 5.3 when the right and the left hand side of the linear system is computed using different techniques.

6.2.2 Error analysis of Rnet and model problem

The error in L^2-norm is indeed larger than that of the corresponding CutFEM solution, however, it does show a slightly converging trend. The convergent trend displays some variation between the different mesh sizes, which may stem from the different fractions of curved and straight cuts occurring. Perhaps a more clear trend would be observed if the training data would only contain planes, for then the intersections would closer resemble a straight cut for a smaller mesh size. However, it is likely that the general error would be larger in that case. It seems plausible that the error is larger for the ellipse and further larger for the flower, since these are expected to constitute more difficult geometries, and we also observe the same trend for CutFEM. The nearly static trend of the energy sub-norm error is not surprising given the low convergence order of the L^2-error. Looking at fig. 5.5, we see that some elements are better approximated, whilst some are worse. Thus, the peaking values of the worse regions must be contributing severely to the error.

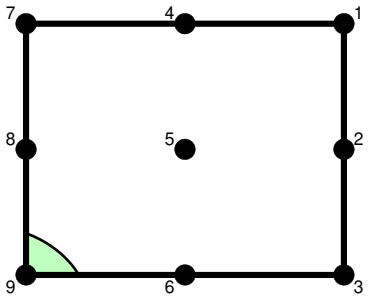
It was shown in the nodal study in fig. 5.6 and the basis function study for different orders in fig. 5.7-5.8 that the majority of the contributions to the error stems from the elements which have minor intersection, and especially the nodes outside of the intersection. This can be further discussed by looking at the corresponding basis function contribution at these nodes. For a one-dimensional example of the basis function, we have the C^1-spline basis functions in $\mathbb{R}^1$ (see fig. 2.3). An example element with small intersection and an illustration of an approximate corresponding spline basis function contribution of node 1 is displayed in fig. 6.1. Node 1 is the furthest from the intersection and has therefore little contribution, which is illustrated by the red fill to $x = \delta$ under the spline function in the plot.

The spline function between 0 and δ can be approximated as $\varphi_1 \approx x^2$ for node 1. Recalling the mass matrix entries on an element K as expressed in eq. 2.48, we may thus approximate the contribution to node 1

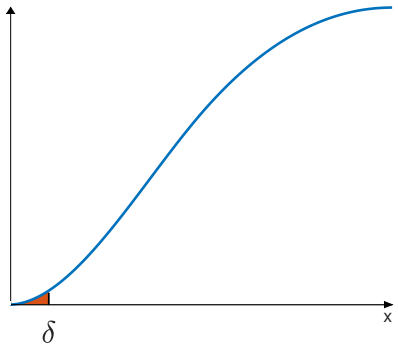
$$M_{11}^K = \int_{K\cap\Omega} \varphi_1\varphi_1 dxdy \approx \int_0^\delta x^4 dx = \frac{\delta^5}{5} \tag{6.2}$$

which rapidly approaches zero when $\delta \to 0$. It is for this reason the contribution at node 1 becomes difficult to approximate, as well as the matrix entries corresponding to M_{1i}^K for $i = 2,..,9$.

Following from above discussion, a potential remedy may be proposed. Since the error weights were applied element-wise in this study, certain nodes (such as node 1) were not especially focused upon and instead the 45 values in the output were treated equally despite



(a) Element with minor intersection (Lagrange element used for graphical illustration).



(b) Spline contribution of basis function for node 1 along one coordinate axis. $0 < \delta << 1$

Figure 6.1: Element with minor intersection and an one dimensional illustration of an approximate corresponding spline basis function contribution for node 1.

the error contribution of node 1 being remarkably smaller than that of the other nodes (due to its lower magnitude). One method of solving this is to apply error weights on all $45N$ values (for N denotes the number of samples). To do this, each mass matrix entry has to be weighted individually in relation to its basis function contribution. Once again recalling the mass matrix expression in eq. 2.48 and applying the Cauchy-Schwarz inequality, we get an upper bound

$$M_{ij}^K = \int_{K\cap\Omega} \varphi_i \varphi_j dxdy \leq \left(\int_{K\cap\Omega} \varphi_i dxdy\right)^{1/2} \left(\int_{K\cap\Omega} \varphi_j dxdy\right)^{1/2} \tag{6.3}$$

thus the mass matrix values would be bounded below one for the following division

$$\frac{M_{ij}^K}{\left(\int_{K\cap\Omega} \varphi_i dxdy \int_{K\cap\Omega} \varphi_j dxdy\right)^{1/2}} \leq 1 \tag{6.4}$$

which would impose a relative measure on the corresponding nodal MSE. Thus, the nodal weights can be determined according to

$$W_{ij} = \frac{1}{w_i w_j} = \frac{1}{\left(\int_{K\cap\Omega} \varphi_i dxdy \int_{K\cap\Omega} \varphi_j dxdy\right)^{1/2}} \tag{6.5}$$

for $W \in \mathbb{R}^{N\times 45}$ (after symmetries in the mass matrix and weights have been removed), and applied on the data set for each element.

A quick comparison between the new and the old weights can be performed for the three relevant cases; when $i = j = 1$, when $i = 1, j \neq 1$, and when $i \neq 1, j \neq 1$. Following the same reasoning as above, we can compute an approximate measure of proportion. Doing so, we first approximate the mass matrix entries as in 6.2, yet we integrate over values up to 1 to to get the different spline situations as is present for different nodes at the element. This yields $M_{1j}^K \propto \delta^3$ and $M_{ij}^K \propto 1$ for $i, j \neq 1$. We can then compute the new weights using eq. 6.5.

Above reasoning yields the results proportional to the values in tab. 6.1 which carries onto the MSE. The suggested weights thus indicate that the remarkably low values related to node

1 will be somewhat dampened whilst other values will not be affected. We see that the effect is greatest upon M_{11}^K, but there is also slightly smaller effect on the matrix entries related to node 1, $M_{1,j\neq 1}^K$. In fig. 5.6 we noticed that the largest error stemmed from M_{11}^K whilst $M_{1,j\neq 1}$ also contributed with some difficulties. These errors would likely be reduced proportionally, since the new weights offer stability primarily for these two cases.

Table 6.1: Proportionality between weighted local mass matrix entries and element intersection as illustrated in fig. 6.1 for two types of weights.

	Area weights	Basis weights
$W \cdot M_{11}^K$	$\propto \delta^4$	$\propto \delta^2$
$W \cdot M_{1,j\neq 1}^K$	$\propto \delta^2$	$\propto \delta$
$W \cdot M_{i\neq 1,j\neq 1}^K$	$\propto 1$	$\propto 1$

6.3 Comments on the time consumption

We see that the computational speed is increased for both methods in comparison to CutFEM by looking at fig. 5.9. This indicates that both networks can be used individually depending on what task is to be performed. It also suggest that a combination of the networks will reduce the time significantly, in comparison to CutFEM.

The difference is especially prominent for Rnet which shows that the removal of the inner quadrature loop over the cut elements has high impact on the speed of the method. Cnet is not as fast as the Rnet algorithm, however, this may stem from some of the required sub-discretisations which had to be included in the matrix assembly for the script to be compatible with other, already existing scripts. Therefore, the time consumption can likely be further reduced by optimising some of the existing scripts to better suit the new method. It has been mentioned previously that a repeated discretisation of the domain is indeed time consuming, which partially comes into effect in the current Cnet method.

6.4 Conclusion

This study was performed with the aim of reducing the computational time for CutFEM by replacing quadrature on cut elements with two neural networks; one classification network for identifying the cut elements, and one regression network to approximate the element matrix contributions of the cut elements. The main findings are:

- The time consumption for the matrix assembly showed a significant decrease for the modified CutFEM in comparison to the conventional method.
- The classification network showed a 100% success rate for two elementary domains, although it displayed some difficulties for a domain with a non-trivial geometry representation.

- The regression network showed promising results for elements with a sufficient intersection of the domain, however, elements with minor intersection contributed majorly to the error. This was deduced to stem from the nodal contributions furthest from the domain intersection. A modification of the error weights as used in the training is proposed as a potential remedy to this instability.

While there are some loose ends that need to be addressed, using a network to replace complicated and time-consuming quadrature rules for cut elements shows great promise for the right applications - where computational speed is essential and some loss of accuracy can be tolerated.

BIBLIOGRAPHY

[1] Dmytro Shulga, Oleksii Morozov, Volker Roth, Felix Friedrich, and Patrick Hunziker.
Tensor b-spline numerical methods for pdes: a high-performance alternative to fem, 2019.
URL https://arxiv.org/abs/1904.03057.

[2] Susanne Claus and Pierre Kerfriden.
A cutfem method for two-phase flow problems.
Computer Methods in Applied Mechanics and Engineering, 348:185–206, 2019.

[3] Erik Burman, Susanne Claus, Peter Hansbo, Mats G. Larson, and André Massing.
Cutfem: Discretizing geometry and partial differential equations.
International Journal for Numerical Methods in Engineering, 104(7):472–501, 2015.
doi: https://doi.org/10.1002/nme.4823.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.4823.

[4] Peter Hansbo, Mats Larson, and Karl Larsson.
Cut finite element methods for linear elasticity problems.
03 2017.

[5] Stéphane Bordas, Erik Burman, Mats Larson, and Maxim Olshanskii.
Geometrically Unfitted Finite Element Methods and Applications Proceedings of the UCL Workshop 2016: Proceedings of the UCL Workshop 2016.
01 2017.
ISBN 978-3-319-71430-1.
doi: 10.1007/978-3-319-71431-8.

[6] Erik Burman.
Ghost penalty.
Comptes Rendus Mathematique - C R MATH, 348, 11 2010.
doi: 10.1016/j.crma.2010.10.006.

[7] J. Nitsche.
Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind.
Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg, 36:9–15, 07 1971.
doi: 10.1007/BF02995904.

[8] Peter Hansbo.
Nitsche's method for interface problems in computational mechanics.
GAMM-Mitteilungen, 28(2):183–206, 2005.
doi: 10.1002/gamm.201490018.
URL `https://hal.archives-ouvertes.fr/hal-01338133`.

[9] Andre Massing, Mats Larson, and Anders Logg.
Efficient implementation of finite element methods on non-matching and overlapping meshes in 3d.
SIAM Journal on Scientific Computing, 35:C23–C47, 01 2013.

[10] Maxim Olshanskii and Danil Safin.
Numerical integration over implicitly defined domains for higher order unfitted finite element methods.
Lobachevskii Journal of Mathematics, 37, 01 2016.
doi: 10.1134/S1995080216050103.

[11] Zeng Zhe-Zhao, Wang Yao-Nan, and Wen Hui.
Numerical integration based on a neural network algorithm.
Computing in Science Engineering, 8(4):42–48, 2006.
doi: 10.1109/MCSE.2006.73.

[12] Steffan Lloyd, Rishad A. Irani, and Mojtaba Ahmadi.
Using neural networks for fast numerical integration and optimization.
IEEE Access, 8:84519–84531, 2020.
doi: 10.1109/ACCESS.2020.2991966.

[13] Boram Yoon.
A machine learning approach for efficient multi-dimensional integration.
Scientific Reports, 11:18965, 09 2021.
doi: 10.1038/s41598-021-98392-z.

[14] Mats Larson and Fredrik Bengzon.
The Finite Element Method: Theory, Implementation, and Applications, volume 10.
01 2013.
ISBN 978-3-642-33286-9.
doi: 10.1007/978-3-642-33287-6.

[15] Karl Larsson, Stefan Kollmannsberger, Ernst Rank, and Mats G. Larson.
The finite cell method with least squares stabilized nitsche boundary conditions, 2021.

[16] Erik Burman and Peter Hansbo.
Fictitious domain finite element methods using cut elements: Ii. a stabilized nitsche method.

Applied Numerical Mathematics, 62(4):328–341, 2012.
ISSN 0168-9274.
doi: https://doi.org/10.1016/j.apnum.2011.01.008.
URL https://www.sciencedirect.com/science/article/pii/S0168927411000298.
Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010).

[17] Tobias Jonsson, Mats Larson, and Karl Larsson.
Graded parametric cutfem and cutiga for elliptic boundary value problems in domains with corners.
12 2018.

[18] Emad A. M. Andrews Shenouda.
A quantitative comparison of different mlp activation functions in classification.
pages 849–857, 2006.

[19] Kenneth Levenberg.
A method for the solution of certain non-linear problems in least squares.
Quarterly of Applied Mathematics, 2(2):164–168, 1944.
ISSN 0033569X, 15524485.
URL http://www.jstor.org/stable/43633451.

[20] Donald W. Marquardt.
An algorithm for least-squares estimation of nonlinear parameters.
Journal of the Society for Industrial and Applied Mathematics, 11(2):431–441, 1963.
ISSN 03684245.
URL http://www.jstor.org/stable/2098941.

[21] Marina Adriana Mercioni and Stefan Holban.
The most used activation functions: Classic versus current.
In *2020 International Conference on Development and Application Systems (DAS)*, pages 141–145, 2020.
doi: 10.1109/DAS49615.2020.9108942.

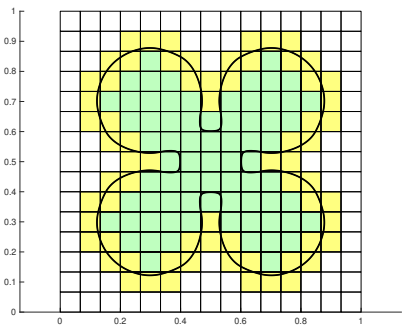
[22] Sagar Sharma, Simone Sharma, and Anidhya Athaiya.
Activation functions in neural networks.
towards data science, 6(12):310–316, 2017.

APPENDIX A

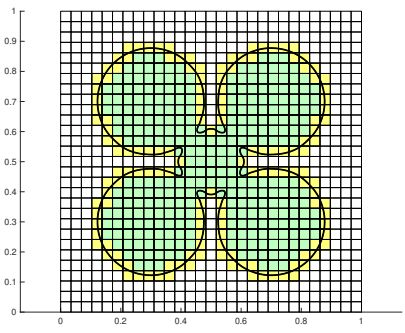
COMPLIMENTARY RESULTS AND REFERENCES

A.1 Classification on flower domain

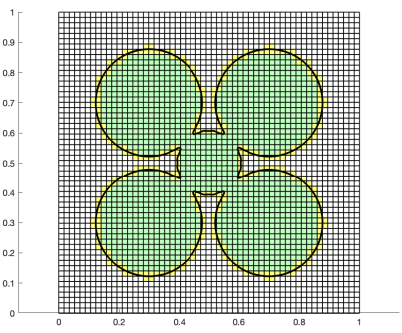
In fig. A.1 we see the classification of the elements on the flower geometry as approximated by Cnet.



(a) $h = 0.1$.



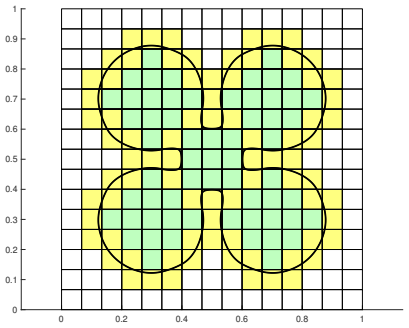
(b) $h = 0.05$.



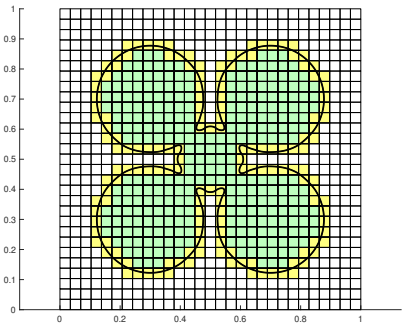
(c) $h = 0.025$.

Figure A.1: Element classification of Cnet with different mesh sizes on the flower geometry. White elements are classified as exterior, yellow as cut and green as interior.

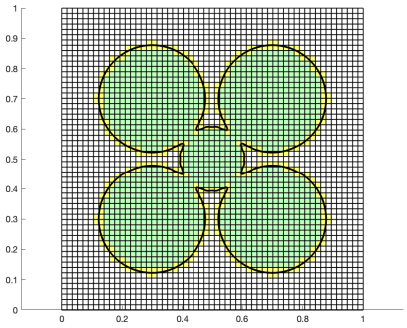
In fig. A.2 we see the corresponding classification as performed using quadrature. The misclassifications of Cnet appear to only occur for the innermost elements where the circles coincide.



(a) $h = 0.1$.



(b) $h = 0.05$.



(c) $h = 0.025$.

Figure A.2: Element classification using quadrature with different mesh sizes on the flower geometry. White elements are classified as exterior, yellow as cut and green as interior.

A.2 Analytical references

Fig. A.3 displays the analytical solution of eq. 4.6 on an ellipse of mesh size $h = 0.1$.

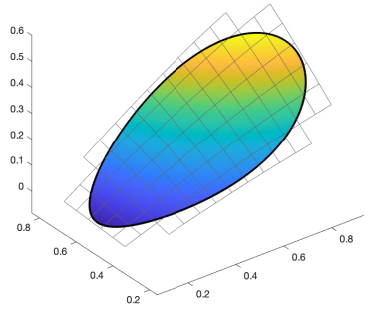
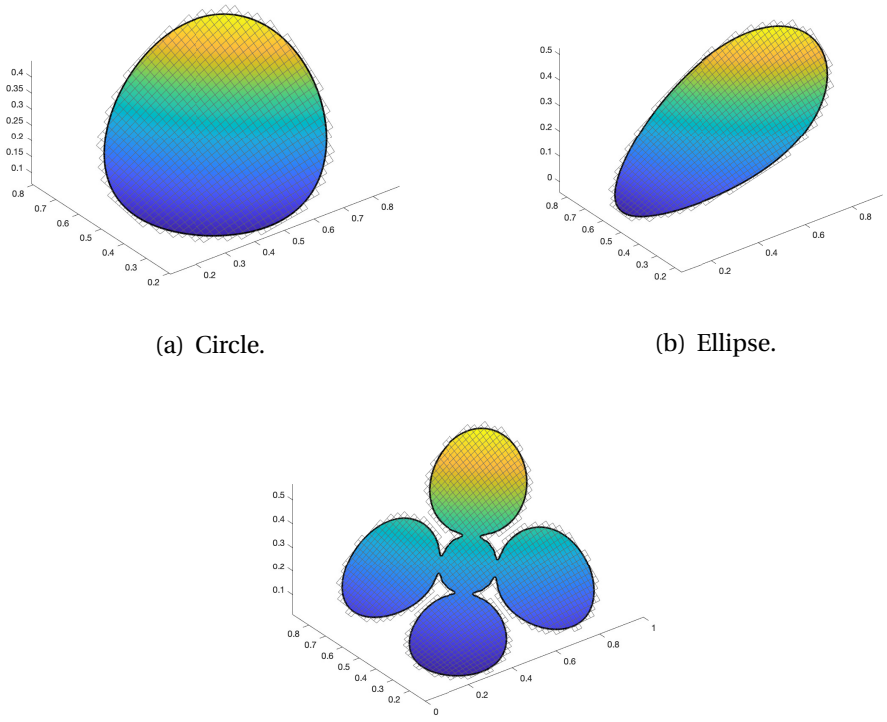


Figure A.3: Analytical solution eq. 4.6 on an ellipse with mesh size $h = 0.1$.

Fig. A.4 displays the analytical solution eq. 4.6 on the circle, ellipse and flower with a mesh size of $h = 0.025$.



(a) Circle.

(b) Ellipse.

(c) Flower.

Figure A.4: Analytical solution eq. 4.6 on three geometries with mesh size $h = 0.025$.