TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Mathematics Institute of Scientific Computing

C^1 Finite Elements for Dune

Maik Porrmann

Master Thesis

to achieve the academic degree

Master of Science (M.Sc.)

First referee

Prof. Dr. Axel Voigt

Second referee

Dr. Simon Praetorius

Supervisor

Prof. Dr. Axel Voigt

Submitted on: 15th June 2022

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Mathematics Institute of Scientific Computing

Abstract

We present a DUNE module that provides three new types of finite elements, namely the cubic Hermite element on simplices, the quadratic Morley and the quintic Argyris triangle. They give DUNE the capability to formulate conforming methods for fourth order problems in one and two dimensions. We detail out, how they handle the lack of affine equivalence while still being usable on the reference element, and discuss their positioning within the DUNE framework. Additionally, a particular focus lies on the possibility to strongly enforce essential boundary conditions. Including these elements in DUNE routines and higher level discretization modules solely requires an adaptation of the global interpolation routine in order to allow the evaluation of derivatives as degrees of freedom and a minor adaption in the treatment of boundary conditions. We verify that the implemented elements attain their theoretically predicted convergence rate for a series of numerical experiments, ranging from second order elliptic problems to the minimization of a von-Kármán energy.

Statement of authorship

I hereby certify that I have authored this document entitled C^1 *Finite Elements for Dune* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 15th June 2022

Maik Porrmann

Digital unterschrieben von Maik Porrmann
Datum: 2022.06.15 11:37:54 +02'00'

Maik Porrmann

Contents

1 Introduction

Finite element methods have seen huge success since their framework has been developed in the 1970s. The theory developed back then often covered both finite elements of Lagrange type and those of Hermite type and authors listed both types as almost equal choices, see for example [Ciarlet, P G, 1978]. In nowadays literature and especially software however, elements of Hermite type, that is finite elements that involve derivatives as degree of freedoms, have almost vanished. Reasons for this include the higher polynomial degrees, complicated implementation of boundary conditions, and the fact that modern finite element software typically only works on the reference element. Due to the lack of affine equivalence (in a strict sense) of finite elements of Hermite type, this is not straightforwardly possible. For higher order problems, where the C^1-continuity of the discrete spaces is required for a conforming method, multiple other approaches within the finite element framework have been developed. In particular, finite elements of class C^1 can be avoided by rewriting the higher order problem as multiple second order problems, which leads to a mixed method, or by means of the C^0 interior penalty approach, analyzed in [Brenner and Sung, 2005], which penalizes the jumps in normal derivatives across the facets of the triangulation. Even though one does not obtain a strongly differentiable solution, the method still yields optimal orders of convergence. In recent years, virtual elements methods have been developed, which offer easier possibilities to implement differentiable spaces, see [Brezzi and Marini, 2013]. Despite those well working alternatives, the class of C^1 finite elements still provides a viable approach, and in principle finite element frameworks can only benefit from providing them in addition to the typically available range of C^0 elements. The present work aims to close this gap for the finite element toolbox DUNE. It was motivated and in fact heavily relies on [Kirby, 2018, Kirby and Mitchell, 2019], who proposed a clean and efficient way to implement finite elements of Hermite type and implemented them in the FInAT framework.

We present a DUNE module named `dune-c1elements`. It is designed as an extension module to `dune-functions`, has no additional dependencies, and is available via git (https://gitlab.dune-project.org/maik.porrmann/dune-c1elements).

The module contains the implementations for three types of finite elements, namely the cubic Hermite element, for one- to three-dimensional simplices, the Morley triangle and the Argyris triangle. Of those, only the one-dimensional Hermite element and the Argyris triangle actually form finite element spaces which are conforming in H^2. The Morley element provides a nonconforming discretization for problems posed in H^2, but the two- and three-dimensional Hermite elements do not. While these are implemented for the sake of completeness, and will in part be analyzed for second order problems in Chapter 5, the primary focus of this work is on the Argyris triangle.

The implemented elements fulfill major parts of the DUNE interface for finite elements. In particular, this means that they can be used straightforward in almost any assembling routine, which is written for objects implementing DUNE's interface. In contrast to most of the other finite elements so far implemented in DUNE, finite elements of Hermite type are generally not affine equivalent. Because the approach here is based on the construction of an affine equivalent finite element on the reference simplex for each physical finite element, our elements can be used in routines that assemble integrals on the reference simplex. However, some routines might use caching throughout the iteration over the triangulation to speed up the evaluation of basis functions. Those approaches do not work with our finite elements, since each finite element in a family has its own and different reference finite element. Furthermore, the implemented nodal interpolation operator departs from the DUNE interface, since it evaluates derivatives. Typically in DUNE, interpolation of a function object, that only implements evaluation, but no derivatives, should be possible. For finite elements of Hermite type, this is not trivial, as they include derivatives as degrees of freedom by definition. Some alternative ways for Hermite interpolation have been proposed to overcome this, see for example [Girault and Scott, 2002], but so far only the classical nodal interpolation operator is implemented. This means in particular, that the standard interpolation routines for global interpolation into the finite element spaces have to be amended, if for example the Argyris element is to be used by the numerous high level discretization modules in the DUNE universe.

Strongly enforcing boundary conditions is a delicate topic for finite elements of Hermite type, and among the reasons for their little usage. In order to provide a convenient interface for this, we implement a version of the Hermite and Argyris elements that features only tangential or normal derivatives as degrees of freedom. This allows the creation of subspaces that obey homogeneous boundary conditions, as well as a straightforward interpolation of inhomogeneous boundary conditions by a subset of boundary degrees of freedom. In this implementation, the choice of tangential and normal degrees of freedom only depends on the grid, with the only exception being non-rectangular corners, where a change from clamped to free boundary conditions happens. The cost paid for this possibility to strongly enforce boundary conditions comes in terms of a strict interface for boundary interpolation, which requires the boundary data to be stated in terms of a single function. While this might be inconvenient in some situations, for example inhomogeneous clamped conditions, in other situations it might not cause problems. At any case, weakly enforcing boundary conditions remains an option as well.

This thesis is structured as follows. In Chapter 2 we give a general introduction to the core ideas of finite element methods, introducing the necessary definitions and general ideas. Chapter 3 reviews the transformation theory for finite elements from [Kirby, 2018] and proposes *grid aligned* variations for finite elements of Hermite type, which allow strong enforcement of boundary conditions under certain conditions. Afterwards, some aspects and interfaces of the DUNE framework are presented in Chapter 4 and the implementation of the finite elements is discussed. The last two chapters contain numerical experiments. Those in Chapter 5 verify the expected rate of convergence for elliptic problems, whereas in Chapter 6 the Argyris element is used to minimize a von-Kármán like energy.

1.1 Preliminaries and Notation

Before we dive into the concrete topic of this work, we list a few concepts and definitions that will be used throughout.

The problems considered in this thesis will be posed over a *domain* Ω. By that, we mean an open, Lebesgue-measurable subset of $\mathbb{R}^d$, whose interior is non-empty. The finite element methods discussed will be based on dividing Ω into *simplices*. Here a simplex in d dimensions is the closed convex hull of $d+1$ vertices $\{ v_i \}_{i=0,\dots,d}$, where the *edge* tangentials $\{ v_i - v_0 \}_{i=1,\dots,d}$ are linear independent. A one-dimensional simplex is an line segment, while a two-dimensional simplex is a *triangle*. A *triangulation* of Ω is a finite set of simplices $\{ T_i \}$ such that

1. The pairwise intersections of the interiors of the simplices are empty,
2. the pairwise intersections of the simplices are either empty or contain exactly one complete subsimplex,
3. and the union of simplices is the closure of Ω.

Most examples will be given in two dimensions. A two-dimensional simplex is called a *triangle*. For $i \in 0, 1, 2$ we will denote the vertices of a triangle with v_i, its edges with e_i, where e_i is the opposed to v_i and has the midpoint m_i, the unit tangential τ_i and outer unit normal vector ν_i.

Furthermore, we use the following notations throughout this work:

- $\mathbb{P}_k(\Omega)$ denotes the space of polynomials of degree k over Ω.
- $\mathrm{W}^{k,p}(\Omega)$ denote the usual Sobolev spaces of p–integrable functions whose weak derivatives up to order k are also p–integrable. The corresponding Hilbertspaces are denoted by $\mathrm{H}^k(\Omega) := \mathrm{W}^{k,2}(\Omega)$. Furthermore, $\mathrm{H}^{\mathrm{k}}_0 \subset \mathrm{H}^{\mathrm{k}}$ is the subspace of functions with derivatives up to order $k-1$ vanishing on $\partial\Omega$.
- The scalarproduct of $v, w \in \mathbb{R}^n$ is denoted by $v \cdot w$, the induced euclidian norm by $|v|$.
- The L^2-scalar product is denoted by $(\cdot, \cdot)$, the induced norm by $\|\cdot\|$.

- The scalar product on $\mathrm{W}^{k,p}(\Omega)$ is denoted by $(\cdot,\cdot)_{k,p,\Omega}$, for the induced norm we write $\|\cdot\|_{k,p,\Omega}$. For the Hilbertspaces $\mathrm{H}^k(\Omega)$ we simply write $(\cdot,\cdot)_{k,\Omega}$ and $\|\cdot\|_{k,\Omega}$ and if obvious from context we will drop the Ω subscript.
- For a normed space V, V' denotes its topological dual space, that is, the space of continuous linear functionals.
- The space $\mathrm{C}_\mathrm{b}^\mathrm{k}$ is the space of functions which have bounded strong derivatives up to order k.

2 Finite Element Methods

2.1 A Short Introduction to Finite Element Methods

This section gives an informal overview over ideas and routines in finite element methods. It is by no means exhaustive or complete. However, it recapitulates some important concepts and points to various topics discussed in later sections.

2.1.1 Variational Problems

Finite element methods are typically used to solve *variational problems*, here given in an abstract form:

$$\begin{aligned} &\text{Find } u \in V \text{ such that} \\ &a(u, v) = b(v) \quad \forall\, v \in V, \end{aligned} \tag{2.1}$$

for some bilinear form a and some linear form b defined on a normed space V. Such a variational problem is typically derived from a *boundary value problem*, that is a partial differential equation with a set of boundary conditions, or from an energy minimization problem.

Definition 2.1 (Continuity of (bi) linear forms)**.** A linear form b is continuous on V iff

$$\exists C_1 > 0\colon\ |b(v)| \le C_1 \|v\|_V \quad \forall\, v \in V.$$

A bilinear form a is continuous on $V \times W$ iff

$$\exists C_2 > 0\colon\ |a(v, w)| \le C_2 \|v\|_V \|w\|_W \quad \forall\, v \in V,\, \forall w \in W.$$

Definition 2.2 (Coercivity)**.** A bilinear form a is coercive on V (or V-elliptic) iff

$$\exists \alpha > 0\colon\ |a(v, v)| \ge \alpha \|v\|^2 \quad \forall\, v \in V.$$

The celebrated Lax-Milgram lemma is the main result in solution theory for variational problems and given below.

Lemma 2.1 (Lax Milgram)**.** *Let V be a real Hilbertspace, $a\colon V\times V \to \mathbb{R}$ a continuous and coercive bilinear form and $b\colon V \to \mathbb{R}$ a continuous linear form. Then* (2.1) *has a unique solution in V.*

2.1.2 Galerkin Methods

The key idea of numerous numerical methods is to "approximate" V by a discrete space V_h and to solve the resulting discrete problem. Strictly speaking, in (2.1) we have two spaces, one for the solution u, called the *trialspace*, and one for the testfunctions v, called the *testspace*. While one can use different discrete spaces, we will focus on the classical *Galerkin method*, where both spaces are approximated by the same discrete space V_h. We distinguish between discrete spaces, that are subspaces of the continuous test space (the *conforming* case) and those that are not (the *nonconforming* case). For the latter, but also to include the effects of numerical methods like quadrature, one has to define modifications the bilinear and linear form, denoted here by $a_h(\cdot,\cdot)$ and $b_h(\cdot)$.

The *discrete problem* then reads:

$$\begin{aligned}&\text{Find } u_h \in V_h \text{ such that}\\ &a_h(u_h,v_h) = b_h(v_h) \quad \forall\, v_h \in V_h.\end{aligned} \tag{2.2}$$

It is intuitively clear, that the error between the true solution u and the discrete solution u_h depends on the "approximation quality" of V by V_h as well as on the relation between a, b and a_h, b_h. This is formalized in the two following lemmata, which give estimates for the error between the discrete solution and the continuous solution.

For the conforming case $V_h \subset V$, it follows that a and b are well-defined on V_h, so, when neglecting quadrature, a_h is defined by $a_h(v_h,w_h) = a(v_h,w_h)$ and b_h is defined by $b_h(v_h) = b(v_h)$ for all $v_h, w_h \in V_h$.

Lemma 2.2 (Céa's Lemma)**.** *Let V and $V_h \subset V$ be real Hilbertspaces, $a\colon V\times V \to \mathbb{R}$ a continuous and coercive bilinear form, $b\colon V \to \mathbb{R}$ a continuous linear form, u the solution to* (2.1) *and u_h the solution to* (2.2)*.*
Then

$$\|u-u_h\|_V \le \underbrace{\frac{C_2}{\alpha}\inf_{v_h\in V_h}\|u-v_h\|_V}_{\textit{discretization error}}.$$

In the nonconforming case, where $V_h \not\subset V$, a,b and $\|\cdot\|_V$ are not a priori well-defined for $v_h \in V_h$. The discrete problem is formulated in terms of a_h, b_h and a mesh-dependent norm $\|\cdot\|_h$, which have to fulfill the following conditions:

- $\|\cdot\|_h : (V+V_h) \to \mathbb{R}$ is a norm on $V_h + V \coloneqq \{\, v+v_h : v\in V, v_h \in V_h \,\}$.
- $a_h\colon (V+V_h)\times(V+V_h) \to \mathbb{R}$ and $b_h\colon V+V_h \to \mathbb{R}$ must agree with a and b on V.

- Additionally, a_h has to be *uniformly* coercive on V_h, that is the coercivity constant α has to be independent of h, and a_h and b_h have to uniformly continuous on $V + V_h$.

Lemma 2.3 (Strangs second Lemma)**.** *With the conditions stated above we have that for u being the solution to* (2.1) *and u_h being the solution to* (2.2)*:*

$$\|u - u_h\|_h \leq \underbrace{(1 + \frac{C_2}{\alpha}) \inf_{v_h \in V_h} \|u - v_h\|_h}_{\text{discretization Error}} + \underbrace{\frac{1}{\alpha} \sup_{v_h \in V_h, \|v_h\|_h \neq 0} \frac{|a_h(u, v_h) - b_h(v_h)|}{\|v_h\|_h}}_{\text{consistency error}}. \tag{2.3}$$

2.1.3 Construction of Finite Element Spaces

So far, we have considered general Galerkin methods. A finite element method is a Galerkin method, where the discrete space V_h is constructed by means of *finite elements*.
Let $\mathcal{T}_h$ be a triangulation of the domain Ω. Finite elements define a discrete space P_T, usually a (sub)space of polynomials up to some degree, over each element $T \in \mathcal{T}_h$. The detailed definition and examples are given in Section 2.2. With the help of those local spaces one constructs the *global basis*, throughout this work denotet by Φ, of a discrete space defined on the whole domain Ω. The choice of finite elements and the details of construction determine the global properties of the obtained global space. In particular, a standard choice are finite elements of *Lagrange type*, where only global C^0 continuity is enforced. As a consequence, the obtained global spaces are not strongly differentiable over the edges of $\mathcal{T}_h$. The focus of the present work, however, lies on finite elements that yield subspaces of $\mathrm{C}^1(\Omega)$. For variational problems derived from a fourth order boundary value problem, this is a necessary condition in order to stay in the conforming regime.

Section 2.3 is dedicated to the construction of *finite element spaces* and the ensuing global properties are discussed in more detail. For this introductory section it suffices to state, that we obtain a discrete space V_h with dimension n, thus every $v_h \in V_h$ obeys

$$v_h = \sum_{i=1}^{n} v_i \phi_i, \tag{2.4}$$

where $v_i \in \mathbb{R}$ and $\Phi \coloneqq \{\, \phi_i \,\}_{i=1,\dots,n}$ forms the aforementioned global basis of V_h.

2.1.4 From a Discrete Variational Problem to an Algebraic Problem

Using (2.4), the discrete problem (2.2) can be written as

$$\text{Find } u_h \in V_h \text{ such that}$$
$$a_h(u_h, v_h) = a_h\left(\sum_{j=1}^{n} u_j\phi_j, \sum_{i=1}^{n} v_i\phi_i\right) = b_h\left(\sum_{i=1}^{n} v_i\phi_i\right) = b_h(v_h) \quad \forall\, v_h \in V_h,$$

where u_i is the coefficient of u_h to the i-th basisfunction and v_i accordingly. By the linearity of a_h and b_h we obtain an algebraic problem:

$$\begin{aligned}&\text{Find } (u_j)_{j=1,\dots,l} \in \mathbb{R}^l \text{ such that}\\ &\sum_{j=1}^{l} a_h(\phi_j, \phi_i)u_j = b_h(\phi_i) \quad \text{for } i = 1, \dots, l,\end{aligned}$$

where $l = \dim(V_h)$.

Note that the finite dimensionality of V_h was used to switch from a problem, whose solution is element of a function space, to a problem, whose solution is a vector in $\mathbb{R}^l$. Hence, it is natural to formulate the problem as a system of linear equations, namely

$$\begin{aligned}&\text{Find } u \in \mathbb{R}^n \text{ such that}\\ &\quad Au = b, \\ &\quad A_{ij} := a_h(\phi_j, \phi_i),\\ &\quad\; b_i := b_h(\phi_i).\end{aligned} \tag{2.5}$$

2.1.5 Assembling the Linear Equation System

The matrix entries $A_{ij} = a_h(\phi_j, \phi_i)$ usually have the form of integrals over derivatives up to order q for a partial differential equation of order $2q$. Since the domain of the basisfunctions is usually small in comparison to the domain, the assembled system is sparse. Even though we will not go into the details, it has to be stated, that the integrals are typically approximated by numerical quadrature. Common finite element software works with the *reference paradigm*. Instead of evaluating the integrals over the different triangles $T \in \mathcal{T}_h$, they are transformed onto a reference triangle $\hat{T}$. If one additionally has a *reference finite element*, that is a finite element defined over $\hat{T}$ that is *affine equivalent* to the finite element defined over T, one can calculate the integrals very efficiently. This transformation and the concrete conditions under which it is applicable are discussed in Section 2.4. While this works out of the box for many standard finite element methods, in particular those utilizing Lagrange finite elements (see Example 2.11), we will see, that the classes of finite elements with higher regularity require an additional transformation.

2.1.6 Solving the Linear System

It remains to solve the assembled system. There exists a variety of direct or iterative solvers, many of which were explicitly designed for solving large sparse systems arising from finite element methods. Frequent choices include direct solvers for small sized problems and iterative solvers like the conjugate gradient method for symmetric and the biconjugate gradient stabilized method for non-symmetric problems, both of which are often used together with suitable preconditioners. Among the fasted methods for large systems are multigrid methods, where the solutions of coarser systems are used to approximate the solutions of finer discretizations.
However, this is not the topic of this work and a detailed introduction to this topic is omitted here.

2.2 Finite Elements

This section introduces the concept of finite elements and equivalence relations between them. It mostly provides definitions to be used in later sections. More detailed discussions can be found in [Ciarlet, P G, 1978] and [Brenner and Scott, 2002].

Definition 2.3 (Finite Element)**.** A triplet (T, P, N) is called *finite element*, if

1. $T \subset \mathbb{R}^d$ closed.
2. P is a (usually polynomial) function space defined over T.
3. $N \subset \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T)'$ is a finite set, linear independent and unisolvent on P.

The finiteness of P follows from finiteness of N and its unisolvence on P. The set N is often called *degrees of freedom* or the *node* set of the finite element. Note that it contains general, albeit linear and continuous, functionals. If the degrees of freedom of a finite element solely contain function evaluations at different points, we say it is a *Lagrange* finite element. Their definitions can be given solely in terms of those points, such that one can discuss Lagrange finite elements, without ever explicitly stating the functionals. In literature like [Brenner and Scott, 2002], this is mirrored languagewise, and so the term node often refers to one of those points as the location of the corresponding degree of freedom, rather than the functional itself. In this work however, we do not distinguish between nodes and degrees of freedom, and address the functionals in N by both terms.

Definition 2.4 (Nodal Basis)**.** For a finite element $\left(T, P, N = \{\, n_i \,\}_{i=1,\dots,\dim(P)}\right)$ the basis $\Psi =$

$\{\,\psi_j\,\}_{j=1,\dots,\dim(P)}$ of P fulfilling

$$n_i(\psi_j) = \delta_{ij} \tag{2.6}$$

is called *nodal basis*.

We will refer to (2.6) as *delta property* of Ψ and N. Additionally, for $n \in N$ we say that $\psi \in \Psi$ is the nodal basis function associated with n if $n(\psi) = 1$. The notation ψ_n is used to indicate this association.

Definition 2.5 (Nodal Interpolant)**.** For a finite element (T, P, N), its *nodal interpolant* $\mathcal{I}_N$ is defined as

$$\mathcal{I}_N \colon \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T) \to P, \quad f \mapsto \sum_{n\in N} n(f)\psi_n.$$

2.2.1 Equivalence of Finite Elements

We first define a equivalence relation between finite elements, which are defined on the same element T.

Definition 2.6 (Interpolation equivalent finite elements)**.** Two finite elements are called *interpolation equivalent*, if their nodal interpolants are equal.

In fact interpolation equivalence is such a strong property, that for example in [Ciarlet, P G, 1978] two interpolation equivalent finite elements are said to be the same finite element.

Furthermore, we consider equivalence relations between two finite elements defined on different elements T and $\hat{T}$. To formulate those relations concisely, we introduce an operator that "moves" functions from $\hat{T}$ to T and an operator that "moves" functionals in the other direction.
Consider an invertible map $F \colon T \to \hat{T}$.

Definition 2.7 (Pull-back)**.** The *pull-back* operation of F is defined as

$$F^* \colon \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(\hat{T}) \to \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T), \quad F^*(\hat{f}) = \hat{f} \circ F.$$

Definition 2.8 (Push-forward)**.** The *push-forward* operation of F is defined as

$$F_* \colon \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T)' \to \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(\hat{T})', \quad F_*(n) = n \circ F^*.$$

Definition 2.9 (Affine equivalent finite elements)**.** Two finite elements (T, P, N) and $(\tilde{T}, \tilde{P}, \tilde{N})$ are called *affine equivalent*, if there exists a affine map $F \colon T \to \tilde{T}$ such that the following equivalencies hold in the sense of sets:

1. $F(T) = \tilde{T}$,
2. $F^*(\tilde{P}) = P$,
3. $F_*(N) = \tilde{N}$.

It follows that the pull-backs of the nodal basis map onto the nodal basis of the affine equivalent finite element, that is,

$$F^*(\tilde{\Psi}_{F_*(n)}) = \Psi_n.$$

The application of the above relation is discussed in Section 2.4.

Definition 2.10 (Interpolation-affine equivalent finite elements)**.** Two finite elements (T, P, N) and $(\tilde{T}, \tilde{P}, \tilde{N})$ are called *interpolation-affine equivalent* if there is a finite element $(\tilde{T}, \tilde{P}, \hat{N})$ such that (T, P, N) is affine equivalent to $(\tilde{T}, \tilde{P}, \hat{N})$ and furthermore $(\tilde{T}, \tilde{P}, \hat{N})$ is interpolation equivalent to $(\tilde{T}, \tilde{P}, \tilde{N})$.

Remark 2.1. Note that Definition 2.9 differs for some traditional definitions of affine equivalent finite elements. For example, Ciarlet defines affine equivalence in a way, that explicitly includes finite elements of Hermite type, which here are considered only to be interpolation-affine equivalent. This distinction is convenient for this work, as it coincides with the different treatment of affine and interpolation-affine equivalent finite elements in Chapter 3.

Proposition 2.4. *[Brenner and Scott, 2002]Suppose (T, P, N) and $(T, P, \tilde{N})$ are finite elements. They are interpolation equivalent, if and only if* $\operatorname{span}(N) = \operatorname{span}(\tilde{N})$ *holds in* $\mathrm{C}_\mathrm{b}^{\mathrm{k}\prime}$.

In the following, we will consider families of finite elements over a triangulation of some domain Ω. However, instead of defining completely different finite elements on each triangle, typically, we define a *generic finite element* $\{\,(T, P_T, N_T)\,\}$, where P_T and N_T are expressed in dependency of T. This allows to define the family $\{\,(T, P_T, N_T)\,\}_{T \in \mathcal{T}_h}$ for a triangulation $\mathcal{T}_h$ of Ω. Because they are defined on subsets of Ω, in a sense they live in the physical world, and so the finite elements in such a family are called *physical finite elements*. In contrast to them, we consider a *reference element* $\hat{T}$, which is not considered to be related to Ω, upon which we define the *reference finite element* $(\hat{T}, \hat{P} := P_{\hat{T}}, \hat{N} := N_{\hat{T}})$.

Such a family is said to be an *affine equivalent family*, if each finite element is affine equivalent to the reference finite element, and said to be an *interpolation-affine equivalent family*, if each finite element in the family is interpolation-affine equivalent to the reference finite element.

2.2.2 Examples

In the following we give some examples that characterize different generic finite elements. We will refer to these generic finite elements and their corresponding families throughout the later chapters.

In order to express the degrees of freedom concisely, we follow the notation of (vectors of) functionals in [Kirby, 2018], summarized in Table 2.1. In light of the theory developed there and summarized in Chapter 3 it is helpful to write the degrees of freedom of a finite element as a vector, using block notation, to fix an order. We will do so in the following and keep the order given here throughout this work.

$n \in \mathrm{C}_\mathrm{b}^{\mathrm{k}'}$	$n(f)$	$m \in (\mathrm{C}_\mathrm{b}^{\mathrm{k}'})^d$	$(m_1, \cdots, m_d)$
δ_a	$f(a)$		
δ_a^v	$\partial_v f(a)$	∇_a	(δ_a^x, δ_a^y)
		$\hat{\nabla}_{\hat{a}}$	$(\delta_{\hat{a}}^{\hat{x}}, \delta_{\hat{a}}^{\hat{y}})$
		∇_a^{vw}	(δ_a^v, δ_a^w)
δ_a^{vw}	$\partial_v \partial_w f(a)$	$\triangle_a$	$(\delta_a^{xx}, \delta_a^{xy}, \delta_a^{yy})$
		$\hat{\triangle}_{\hat{a}}$	$(\delta_{\hat{a}}^{\hat{x}\hat{x}}, \delta_{\hat{a}}^{\hat{x}\hat{y}}, \delta_{\hat{a}}^{\hat{y}\hat{y}})$
		$\triangle_a^{vw}$	$(\delta_a^{vv}, \delta_a^{vw}, \delta_a^{ww})$

Table 2.1: Notation for degrees of freedom. Here, a denotes a point in T, $\hat{a}$ denotes a point in $\hat{T}$ and x, y are the coordinate axes in the global coordinate system, while $\hat{x}, \hat{y}$ are the coordinate axes in the reference system.

Example 2.11 (Lagrange Simplex of degree k). The Lagrange finite element (T, P, N) of degree k is given by:

- a simplex T,
- $P = \mathbb{P}_k(T)$,
- $N = (\delta_{z_i})_{i=1,\dots,\dim(P)}$,

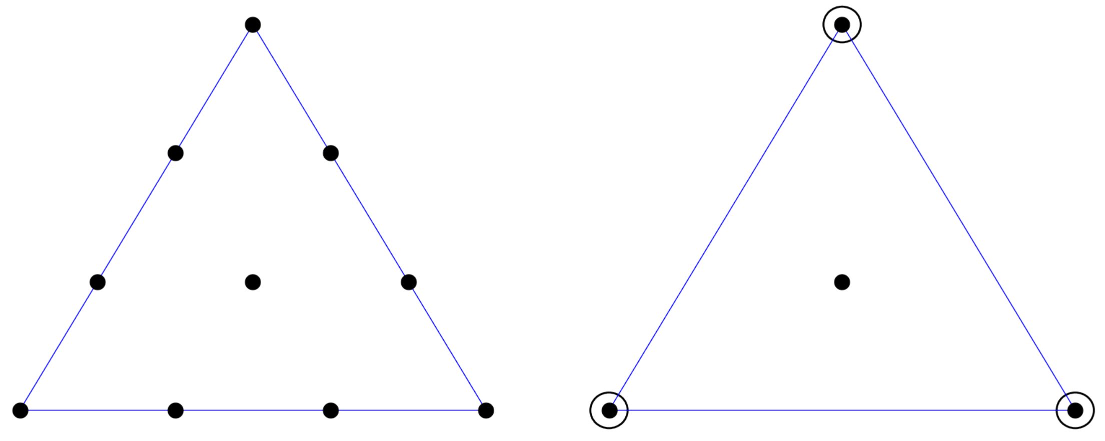
where multiple choices for the locations z_i are possible. The standard choice are the vertices and for higher orders k uniformly distributed points on the boundary of T and the inner of T.

A family of Lagrange finite elements of same degree (and point distribution) defined over the elements of some triangulation $\mathcal{T}_h$ is an affine family.

Example 2.12 (Cubic Hermite Triangle). The cubic Hermite finite element (T, P, N) in two dimensions is given by:

- a triangle T,
- $P = \mathbb{P}_3(T)$,
- $N = (\delta_{v_0}\, \nabla_{v_0}\, \delta_{v_1}\, \nabla_{v_1}\, \delta_{v_2}\, \nabla_{v_2}\, \delta_b)$,

where v_0, v_1, and v_2 are the vertices of T and b is its barycenter.



(a) Lagrange triangle of degree 3

(b) Hermite triangle

Figure 2.1: Cubic Lagrange and Hermite finite elements on a triangle. Dots represent the evaluation of a function value whereas circles represent the evaluation of gradients.

The degrees of freedom of the one-dimensional cubic Hermite element simply consist of the point evaluation and the derivative at each endpoint of its interval, whereas for the three-dimensional element the point evaluations and gradient components at each vertex of the simplex are completed by point evaluations at the barycenter of each facet.

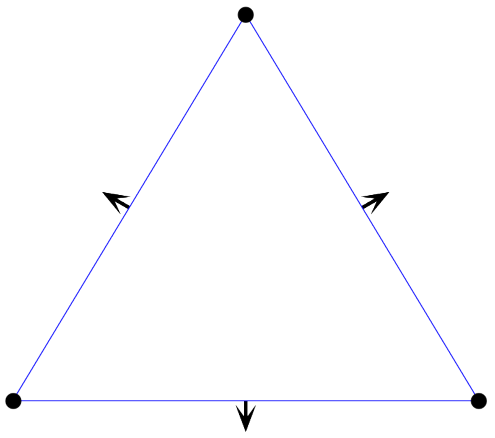
The cubic Hermite finite elements defined over some triangulation $\mathcal{T}_h$ do not form an affine family, as the push-forward does not preserve the length and direction of the derivative degrees of freedom. However, due to Proposition 2.4, they form an interpolation-affine family, since the full gradient is used at each vertex. Hence, for the d-dimensional case, d directional derivatives are mapped to d different but linear independent derivatives, if the mapping is invertible. The Lagrange triangle of degree 3 and the Hermite triangle are depicted in Figure 2.1a and Figure 2.1b.

Example 2.13 (Morley Triangle). The Morley finite element (T, P, N) is given by:

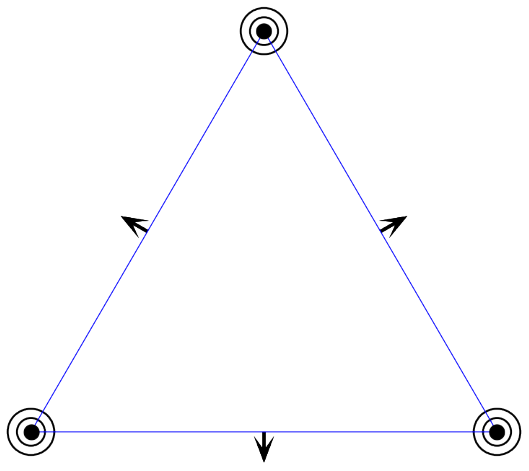
- a triangle T,
- $P = \mathbb{P}_2(T)$,
- $N = \left(\delta_{v_0}\, \delta_{v_1}\, \delta_{v_2}\, \delta_{m_0}^{\nu_0}\, \delta_{m_1}^{\nu_1}\, \delta_{m_2}^{\nu_2}\right)$,

where v_i is the ith vertex, m_i is the midpoint of the ith edge and ν_i is the outer unit normal vector of the same edge.

In contrast to the Hermite element, the Morley element does not form an interpolation-affine family, since at the edge midpoints only the normal derivative is contained in N.



(a) Morley triangle



(b) Argyris triangle

Figure 2.2: Morley and Argyris triangles. Additionally to the symbols from Figure 2.1, here arrows represent the evaluation of the normal derivative and outer circles represent evaluation of the unique entries in the Hessian matrix.

Example 2.14 (Argyris Triangle). The Argyris finite element (T, P, N) is given by:

- a triangle T,
- $P = \mathbb{P}_5(T)$
- $N = (\delta_{v_0}\ \nabla_{v_0}\ \triangle_{v_0}\ \delta_{v_1}\ \nabla_{v_1}\ \triangle_{v_1}\ \delta_{v_2}\ \nabla_{v_2}\ \triangle_{v_2}\ \delta_{m_0}^{\nu_0}\ \delta_{m_1}^{\nu_1}\ \delta_{m_2}^{\nu_2})$,

The Morley triangle and the Argyris triangle are depicted in Figure 2.2a and Figure 2.2b.

For the same reason as for the Morley element, a family of Argyris elements is not an interpolation-affine family. Both elements are only defined for domains in $\mathbb{R}^2$.

Similar to the Lagrange elements, all the mentioned finite elements can be generalized to families, elements of which have higher polynomial degree. However, we will restrict ourselves to the examples above.

2.3 Finite Element Spaces

In this section the construction of finite element spaces over a domain Ω is described as a bottom-up approach by identifying those degrees of freedom in finite element families, that correspond to the same functional in $\mathrm{C}_\mathrm{b}^{\mathrm{k}}{}'(\Omega)$. Combining the nodal basis functions of these degrees of freedom yields bases of discrete spaces, which obey global properties like continuity or differentiability.

Throughout this section we consider a family of finite elements $\{ (T, P(T), N_T) \}_{T\in\mathcal{T}_h}$ with nodal basis Ψ_T, defined over a triangulation $\mathcal{T}_h$ of Ω.

2.3.1 From Finite Elements to Finite Element Spaces

So far, we have had a local perspective on finite elements and their nodal basis functions. In order to provide a global basis, elements of which are defined on the whole domain Ω, we change to a global perspective and extend the domain of the local basis functions Ψ_T by setting them to zero outside their respective element. We will not differentiate notationally between those two views on the basis functions.
As a second (and technically optional) step, degrees of freedom from different finite elements, which are restrictions of the same functional in $\mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(\Omega)'$, are identified and their respective nodal basis functions are combined. Depending on the type of identified degrees of freedom, this step enforces global properties.
The union of (combined) nodal basis functions gives the global basis Φ of the *global finite element space*. Likewise, the union of (identified) degrees of freedom gives the set of *global degrees of freedom M*.

The global finite element space typically is (a subspace of) some space of piecewise polynomials. The examples below refer to this class of spaces:

$$S^{k,m}(\mathcal{T}_h) := \{ v \in \mathrm{C}^m(\Omega) : v|_T \in \mathbb{P}_k(T) \quad \forall T \in \mathcal{T}_h \}, \quad k, m \in \mathbb{N}_0. \tag{2.7}$$

As a special case, $S^{k,-1}(\mathcal{T}_h) := \{ v \in \mathrm{L}^\infty(\Omega) : v|_T \in \mathbb{P}_k(T) \quad \forall T \in \mathcal{T}_h \}$ denotes the space of piecewise polynomials of degree $k \in \mathbb{N}_0$ which are discontinuous over the facets of $\mathcal{T}_h$.

2.3.2 Discontinuous Spaces

Discontinuous spaces form the most simple outcome of the described approach. For these spaces, no nodes are identified, hence the global Basis Φ is the disjoint union of the element bases:

$$\Phi = \dot{\bigcup}_{T\in\mathcal{T}_h} \Psi_T.$$

For example, when using Lagrange finite elements of order k defined over elements T of a triangulation $\mathcal{T}_h$ of Ω, one obtains the global space $\mathrm{S}^{k,-1}(\mathcal{T}_h)$. The spaces obtained are discontinuous over the edges of the underlying triangulation and are hence only subspaces of L^∞. Accordingly, for problems posed in H^1, usage of those spaces results in a nonconforming method. Discontinuous spaces do find some application, but they are not of concern for this work.

2.3.3 Continuous and Differentiable Spaces

Let $\mathcal{N}$ be an indexset, that assigns each degree of freedom of each finite element in the family a unique index. Consider a subset $\mathcal{M}_m \subset \mathcal{N}$ such that $\{ n_i \}_{i \in \mathcal{M}_m}$ contains all those degrees of freedom, that are restrictions of the same global functional $m \in \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(\Omega)'$. These degrees of freedom are considered to be the same, in other words, they are identified with each other. Their associated nodal basis functions $\{ \psi_{n_i} \}_{i \in \mathcal{M}_m}$ can be combined to a single global basis function ϕ_m such that

$$\phi_m|_T = \begin{cases} \psi_n \in \Psi_T, & \text{if } n \in N_T \\ 0, & \text{otherwise} \end{cases} \qquad \forall T \in \mathcal{T}_h, \tag{2.8}$$

where $n := m|_{\mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T)}$ is restriction of m to $\mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T)$ if existent, or the null-functional. For facets shared by two elements, we have possible discontinuities. These are discussed below.

Applying this approach onto all degrees of freedom yields the set of global degrees of freedom

$$M := \{ m \in \mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(\Omega)' : \exists T \in \mathcal{T}_h, m|_{\mathrm{C}^{\mathrm{k}}_{\mathrm{b}}(T)'} \in N_T \} \tag{2.9}$$

and a global basis $\Phi := \{ \phi_m \}_{m \in M}$, which fulfill the delta property by design.

By identifying degrees of freedom, we explicitly enforce some local properties of the resulting function space, in the cases considered here, we enforce continuity or differentiability in the points, where the functionals are located. For example, if we identify all local degrees of freedom that are restrictions of δ_v for some vertex $v \in \Omega$ in the triangulation, then by the delta property all functions in Φ are continuous in v. For many finite elements, those local properties induce global properties as well. Examples are global continuity or differentiability, but also more special properties, like in the case of the Morley triangle, convergence of a nonconforming method for certain problems. Here, we focus on the case of continuity or differentiability. These properties typically hold inside a triangle T, since the elements of $\mathbb{P}_k(T)$ are smooth.
The remaining conditions can usually be seen as an equality of the global basis functions over the facets of neighboring elements, i.e., for two elements T and T' that share a facet f, the conditions for continuity of any global basis function $\phi \in \Phi$ on f can be expressed as

$$\phi|_T(x) = \phi|_{T'}(x) \quad \forall x \in f \quad \forall \phi \in \Phi, \tag{2.10}$$

whereas the conditions for differentiability would read

$$\nabla\phi|_T(x) = \nabla\phi|_{T'}(x) \quad \forall x \in f \quad \forall \phi \in \Phi. \tag{2.11}$$

The first condition is fulfilled, if the global degrees of freedom, that are located on f and hence are shared by the two finite elements, fully specify $p|_f$ for all $p \in P(T) \cup P(T')$, whereas in order to fulfill the differentiability conditions, they have to specify $(\nabla p)|_f$ for all $p \in P(T) \cup P(T')$. The following examples will discuss these properties for the Lagrange, Hermite and Argyris element.

2.3.4 Examples

Example 2.15 (Continuous Lagrange Element). We consider Lagrange finite elements on simplices with a point distribution such that the degrees of freedom located on each facet f are unisolvent on $\mathbb{P}_k(f)$, like the one depicted in Figure 2.1a.

Continuity At every inner edge $f = T_1 \cap T_2$ the two generally different polynomials $v|_{T_1} \in \mathbb{P}_k(T_1)$ and $v|_{T_2} \in \mathbb{P}_k(T_2)$, i.e., the restrictions of some discrete function $v \in V_h$, agree on exactly as many degrees of freedom located on f as needed to fully specify a polynomial of degree k on f. In other words, the combined function v is continuous over f, and since this holds for all facets, v is continuous in Ω.

Differentiability The same argument however, can not be applied for differentiability, since the Lagrangian degrees of freedom do not specify any derivative values. Consequentially, a general $v \in V_h$ is not differentiable over the edges of the triangulation. However, they are weakly differentiable, since they are strongly differentiable on the interior of every triangle T.

Conformity We have $V_h = \mathrm{S}^{\mathrm{k},0}(\mathcal{T}_h) \subset \mathrm{H}^1(\Omega)$ by the arguments above. Let Φ be the basis of V_h. Then $\Phi_0 \coloneqq \Phi \setminus \{\, \phi_m : m \text{ is located on the boundary} \,\}$ is a basis of a discrete subspace $V_h^0 \subset V_h$ that also fulfills $V_h^0 \subset \mathrm{H}_0^1$. Using Lagrange finite elements, we can thus construct discrete conforming spaces for problems posed in H^1 and H_0^1.

Example 2.16 (Cubic Hermite Element).

Continuity Since two neighboring Hermite elements, whose domains share a facet f, agree on the values and gradients in the corners of f, they also agree on the tangential derivative along f in the corners. That is, they agree on four linear independent functionals on f and hence fully determine the values of a polynomial of order 3 on f. Hence, any discrete function constructed by Hermite elements is continuous.

Differentiability In order to be differentiable along f, the two neighboring finite elements additionally have to agree on the normal derivative along f. For the one-dimensional case, this is trivially given, as the normal derivative is just the derivative at f, which is a point. The one-dimensional Hermite element therefore yields a subspace of $\mathrm{C}^1(\Omega)$. For higher dimensions however, the normal derivative along f is a polynomial of order 2, but the two neighboring finite elements only agree on the normal derivative at the corners of f, which makes up for two suitable functionals (for the two-dimensional case). Hence, the discrete function spaces constructed with Hermite elements are not subspaces of C^1.

Conformity The Hermite finite element in 1d can be used to construct subspaces of H^2 and H^2_0. For 2d and 3d, similar as for the Lagrange elements, conforming subspaces for H^1 and, with some modifications also H^1_0, can be constructed. Details on which degrees of freedom on the boundary are to be selected are given in Section 3.3. Note that the created subspaces are strict subspaces of $\mathrm{S}^{3,0}(\mathcal{T}_h)$ for nontrivial triangulations, due to strong differentiability at the vertices of $\mathcal{T}_h$.

Example 2.17 (Argyris Element).

Continuity By similar arguments as before, a discrete function obtained a family of Argyris elements is continuous. Since two neighboring Argyris elements agree on the gradient and the Hessian at the corners of the shared edge e, they also agree on the first and second tangential derivative, which, together with the function value at the corners, makes up for six linear independent functionals fully determining the quintic polynomial along e.

Differentibility In contrast to before, two neighboring Argyris elements also agree on the normal derivative along the shared edge e, since the agree on $\delta^{\nu}_{v_1}$, $\delta^{\nu\nu}_{v_1}$, δ^{ν}_{m}, $\delta^{\nu}_{v_2}$, and $\delta^{\nu\nu}_{v_2}$, where ν is the normal unit vector of e, m the midpoint of e and v_1 and v_2 are its corners.

Conformity In summary, the construction of a finite element space using the Argyris elements yields a subspace of $\mathrm{S}^{5,1}(\mathcal{T}_h)$, since the discrete functions are twice differentiable in the vertices of $\mathcal{T}_h$. It can be used for conforming methods in H^2 as well as $\mathrm{H}^2 \cap \mathrm{H}^1_0$ and H^2_0, again see Section 3.3.

2.4 Consequences of Affine Equivalence

Constructing finite element spaces from affine or affine-interpolation equivalent families has both theoretical and practical advantages.

On the one hand, these properties can be used to obtain error estimates for a particular finite element method. Those error estimates can be found similarly both in the case of an affine family or of an interpolation-affine family. In principle, one starts by constructing a global interpolant $\mathcal{I}_h$, which is equal to the nodal interpolant $\mathcal{I}_N$ of a finite element (T, P, N), when restricted to the corresponding element, i.e., $\mathcal{I}_h|_T = \mathcal{I}_N$. In case of affine equivalent or interpolant affine families, an error estimates for this interpolant can be obtained by an homogeneity argument from the error estimate of the finite element's nodal interpolants. For finite elements, which do not form interpolant affine families, an error estimate for the global interpolant can sometimes be obtained by means of an *almost affine* property. We do not go into the details here, but refer to [Ciarlet, P G, 1978], where in particular the almost affine property of the Argyris triangle is discussed. In combination with Céa's lemma or Strang's second lemma, one can then obtain concrete error estimates for the particular finite element method.

For implementational purposes, the properties of affine equivalent families allow applying the *reference paradigm*. In essence, this means that routines implementing a finite element method should exploit the definition of an affine family of finite elements, by avoiding to actually work on the physical triangles, but rather on the reference triangle using the reference finite element. In particular, in all assembling routines, integrals over T are transformed onto the reference element $\hat{T}$ and evaluated in terms of the reference basis $\hat{\psi}$. For example, consider an entry A_{ij} of the system matrix of a Poisson equation. It will be discussed in more details in the subsequent section. Neglecting boundary terms, we have

$$A_{ij} = \int_\Omega \nabla\phi_j \nabla\phi_i dx = \sum_{T\in\omega_{ij}} \int_T \nabla\phi_j\nabla\phi_i dx = \sum_{T\in\omega_{ij}} \int_T \nabla\psi_{j,T}\nabla\psi_{i,T} dx,$$

where $\omega_{ij} \subset \mathcal{T}_h$ is a set of triangles such that $(\operatorname{dom}\phi_i \cap \operatorname{dom}\phi_j) \subset \omega_{ij}$, $\{\,\phi_i\,\}$ is the global basis and with an abuse of notation we let $\psi_{i,T}$ denote the nodal basis function of the finite element on T that corresponds to ϕ_i. These integrals on single triangles can be transformed to the reference triangle,

$$\begin{aligned} A_{ij} &= \sum_{T\in\omega_{ij}} \int_{\hat{T}} F_T^*(\nabla\psi_{j,T}) F_T^*(\nabla\psi_{i,T}) \left|\det(DF_T)\right| d\hat{x} \\ &= \sum_{T\in\omega_{ij}} \int_{\hat{T}} \left(\hat{\nabla} F_T^*(\psi_{j,T}) DF_T\right)\left(\hat{\nabla} F_T^*(\psi_{i,T}) DF_T\right) \left|\det(DF_T)\right| d\hat{x}, \end{aligned} \tag{2.12}$$

where $F_T\colon \hat{T} \to T$ is an affine mapping with Jacobian DF_T. For affine equivalent families, we have a reference finite element with nodal basis $\hat{\Psi}$ and $F_T^*(\Psi_T) = \hat{\Psi}$ for all $T \in \mathcal{T}_h$, such that (2.12) can be formulated purely in terms of the reference basis and the affine mapping.

Assembling routines typically compute the matrix entries A_{ij} in the above form. To do so, it is necessary, that for every physical finite element (T, P, N) in the family, there exists an affine equivalent finite element defined over the reference element, such that we can evaluate the pullbacks of the physical nodal basis. However, it is not necessary, albeit convenient, that all physical finite elements have the same reference finite element. While the latter only holds for affine equivalent families, the former can be constructed for the Hermite, Morley and Argyris elements as well. Chapter 3 is dedicated to the construction of this affine equivalent finite element.

Lastly we note, that this approach can be generalized to non-affine diffeomorphisms, for example to solve problems define on a surface. In this case, the structure of (2.12) remains the same, and the major question is how to obtain the pullbacks of the physical basis functions.

2.5 Boundary Value Problems

2.5.1 Reformulation as Variational Problem

When given a boundary value problem, that is a partial differential equation with boundary conditions, one typically arrives at the corresponding variational problem, called *weak formulation*, by multiplication with a testfunction and integration over the domain Ω, commonly followed by integration by parts to lower the differentiability requirements of the test- and trialspace. The boundary conditions are incorporated in two different fashions. Loosely speaking, for a PDE of order $2q$, boundary conditions on derivatives up to order q, as well as periodic boundary conditions, are considered *essential* boundary conditions and are encoded in the testspace V, whereas higher order boundary conditions are called *natural* boundary conditions and are encoded in the (bi-)linear forms a and b. The so obtained problem is a variational problem in form of (2.1).
The solution to this variational problem is called *weak solution*, whereas the solution to the corresponding boundary value problem is called *strong solution*. Typically, if the weak solution exists and has the necessary degree of differentiability, it is also a strong solution.

2.5.2 Conforming Treatment of Essential Boundary Conditions

We first consider conforming methods, where the functions in the discrete space fulfill homogeneous boundary conditions. Periodic boundary conditions can be included by different approaches, the maybe most general constructs a conforming discrete space in the spirit of Section 2.3 by identifying the degrees of freedom on the periodic part of the boundary. Homogeneous Dirichlet conditions (or essential conditions on higher derivatives) are encoded in a discrete space $V_{h0} \subset \mathrm{H}_0^{\mathrm{k}}$ by removing the basisfunctions, that are associated to the global functionals needed to specify those conditions. Inhomogeneous essential conditions lead to a affine space $V_c := V_0 \oplus u_c$, where V_0 is the subspace of V that fulfills the corresponding homogeneous conditions and $u_c \in V$ fulfills the inhomogeneous conditions. The variational problem

$$\begin{aligned} &\text{Find } u \in V_c \\ &a(u,v) = b(v) \quad \forall\, v \in V_0 \end{aligned}$$

is rewritten to obtain a problem where test- and trialspace agree:

$$\begin{aligned}&\text{Find } \tilde{u} \in V_0\\ &a(\tilde{u}, v) = b(v) - a(u_c, v) = \tilde{b}(v) \quad \forall\, v \in V_0.\end{aligned}$$

A suitable u_{hc} approximating u_c can be constructed by interpolating the boundary conditions into a suitable set of degrees of freedom, i.e.,

$$u_{hc} = \sum_{m \in M_\Gamma} m(g)\phi_m, \tag{2.13}$$

where g is a function encoding the inhomogeneous condition, called *boundary data*, and $M_\Gamma \subset M$ is chosen such that:

$$V_{h0} \coloneqq V_h \cap V_0 = \operatorname{span}\left(\Phi \setminus \{\, \phi_m : m \in M_\Gamma \,\}\right). \tag{2.14}$$

The resulting discretized problem can be solved by the previously explained methods and the solution to the discrete problem with inhomogeneous conditions is by $u_h = \tilde{u}_h + u_{hc}$, where $\tilde{u}_h$ (respectively u_h) is the discrete approximation of $\tilde{u}$ (respectively u).

This approach in only one of several ways to incorporate essential boundary conditions and not always applicable. In particular, a suitable subset M_Γ is not always available. For example, consider the classical Hermite element and a problem with Dirichlet boundary conditions. To construct a conforming subspace V_{h0} one needs to set the tangential derivative δ_v^τ for a vertex v on the boundary to zero. However, if the grid is not axis aligned, this is not trivial, since only δ_v^x and δ_v^y are elements of M. Hence, including such a boundary conditions requires a more complicated manipulation of the linear equation system. While this is certainly possible, the finite elements presented in Chapter 3 are designed to avoid this by implementing tangential derivative degrees of freedom, thus offering a suitable subset $M_\Gamma \subset M$, and fall into the scope of the method presented above.

Example 2.18 (Diffusion Reaction equation with Dirichlet conditions). The strong form of the Diffusion Reaction equation with all coefficients set to 1 and Dirichlet boundary conditions is given by:

$$\begin{aligned}-\Delta u + u &= f \quad \text{in } \Omega,\\ u &= g \quad \text{on } \Gamma.\end{aligned} \tag{2.15}$$

Multiplication by a test function $v \in \mathrm{C}_\mathrm{c}^\infty(\Omega)$ and integration by parts yields

$$\text{Find } u \in \mathrm{C}^2(\Omega) \text{ such that}$$
$$\int_\Omega \nabla u \nabla v - \underbrace{\int_\Gamma v\,\partial_\nu u}_{=0} + \int_\Omega uv = \int_\Omega fv \quad \forall\, v \in \mathrm{C}_\mathrm{c}^\infty(\Omega) \tag{2.16}$$
$$u = g \quad \text{on } \Gamma.$$

At last, the smoothness requirements are weakened and the inhomogeneous boundary conditions included. We arrive at

$$\text{Find } u \in \mathrm{H}^1_\mathrm{D}(\Omega) \text{ such that}$$
$$\int_\Omega \nabla u \nabla v + \int_\Omega uv = \int_\Omega fv - \int_\Omega \nabla u_D \nabla v - \int_\Omega u_D v \quad \forall\, v \in \mathrm{H}^1_0(\Omega), \tag{2.17}$$

where $\mathrm{H}^1_\mathrm{D}(\Omega) := u_D \oplus \mathrm{H}^1_0(\Omega)$ for some $u_D \in \mathrm{H}^1(\Omega)$ that obeys $u_D = g$ on Γ. Provided, we have a finite element space with a suitable subset of degrees of freedom M_Γ, (2.17) can be discretized as described above.

2.5.3 Nonconforming Treatment of Essential Boundary Conditions

In situations where no suitable set M_Γ is available, one can opt for a nonconforming method to incorporate boundary conditions. As briefly introduced in Section 2.1.2, the idea is to define a modified problem on a discrete space V_h, which is not a subspace of V, in this case because its elements do not fulfill the boundary conditions. Clearly, the solution of the original problem also has to be the unique solution of the modified problem. In Example 2.19, Nitsche's method [Nitsche, 1971] for Dirichlet boundary conditions is considered. It will be numerically investigated in Example 5.2.

Example 2.19 (Nitsche's method for the Poisson equation with Dirichlet conditions). Consider the strong form of the Poisson equation with Dirichlet boundary conditions

$$-\Delta u = f \text{ in } \Omega,$$
$$u = g \text{ on } \Gamma.$$

Its weak formulation is given by:

$$\text{Find } u \in \mathrm{H}^1_\mathrm{D}(\Omega) := \{\, v \in \mathrm{H}^1(\Omega) : v = g \text{ on } \Gamma \,\} \text{ such that}$$
$$\int_\Omega \nabla u \nabla v - \int_\Gamma \partial_\nu u v = \int_\Omega fv. \tag{2.18}$$

If u solves (2.18) it also solves

$$\int_\Omega \nabla u \nabla v - \int_\Gamma \partial_\nu u v + \eta \int_\Gamma (u-g)v = \int_\Omega fv \quad \forall\, v \in \mathrm{H}^1_0(\Omega) \tag{2.19}$$

for some penalization parameter $\eta > 0$. To restore symmetry of the bilinear form, another zero-term is added and the nonconforming problem reads

$$\begin{aligned} &\text{Find } u \in \mathrm{H}^1(\Omega) \text{ such that} \\ &\int_\Omega \nabla u \nabla v - \int_\Gamma \partial_\nu u v - \int_\Gamma (u-g)\partial_\nu v \\ &+\eta \int_\Gamma (u-g)v = \int_\Omega fv \quad \forall\, v \in \mathrm{H}^1(\Omega). \end{aligned} \tag{2.20}$$

2.5.4 Treatment of Natural Boundary Conditions

Natural boundary conditions, given as $B(u) = g$ on Γ with some differential operator B, are enforced by modifying the linear form of the variational problem. Typically the bilinear form contains boundary integrals, where the expression $B(u)$ appears in the argument. Substitution then leads to an integral term, that only depends on v, and can hence be included in a modified linear form $\tilde{b}$.

2.5.5 Boundary Conditions for Fourth Order Problems

For fourth order problems there is a variety of boundary conditions, each given as a combination of essential and natural conditions. Grouping them by their essential part we will consider:

Clamped conditions Both values and normal derivatives of u are specified along Γ. There are no natural conditions.

Simply Supported conditions Only values of u are given as essential boundary conditions. There are natural boundary conditions, the concrete form of which depends on the problem.

Free conditions There are no essential, but only natural boundary conditions.

3 Transforming Finite Elements

This chapter deals with techniques to transform finite elements of non-affine families, such that they can be used via the reference paradigm. It is effectively a summary of [Kirby, 2018], however leaves out the discussion of the Bell element, but gives some details, how one can use the ideas of [Kirby, 2018] to implement finite elements of Hermite type, that allow simple treatment of essential boundary conditions.

3.1 Transformation Theory

We have introduced preliminary definitions in Section 2.2, in particular the general definition of a finite element, as well as the pull-back and the push-forward. We extend these two operators onto sets or vectors of functions, respectively functionals, by elementwise application. For the sake of notational consistency, we write vectors of functions as column vectors and vectors of functionals as row vectors and denote the pullback of the inverse mapping by $F^{-*} := (F^{-1})^*$.

With this notation, we have for a vector N of l' functionals and a vector Φ of l functions,

$$N(\Phi) \in \mathbb{R}^{l \times l'} \text{ with } N(\Phi)_{ij} = n_j(\phi_i)$$

as well as

$$N(M\Phi) = NM(\Phi)$$

for any matrix $M \in \mathbb{R}^{l' \times l}$.

In Definition 2.3, the degrees of freedom are given as elements of the infinite-dimensional dual space $\mathrm{C}_\mathrm{b}^{\mathrm{k}\prime}$. In the following, we will need to distinguish between them and their restriction to element of P. Therefore, for a finite element (T, P, N), and for any $n \in N$, we define $\pi n \in P'$ as restriction of n to P by $\pi n(p) = n(p)$ for all $p \in P$. Again, we vectorize this restriction operator, such that $P' \supset \pi N := \{\pi n : n \in N\}$. Note the equivalence between the statement, that N is unisolvent on P, and the statement, that πN is a basis of P' ([Ciarlet, P G, 1978]).

The basic idea on how to implement non-affine finite elements stems from the three following lemmata.

First we establish the existence of an affine equivalent finite element for every physical finite element.

Lemma 3.1. *Let $F\colon T \mapsto \hat{T}$ be an affine mapping and (T, P, N) be a finite element, then $(\hat{T}, F^{-*}(P), F_*(N))$ is also a finite element and trivially affine equivalent to (T, P, N).*

Proof. Since F is affine, both F^* and F_* are invertible, they preserve vectorspace operations and hence map linear independent sets onto linear independent sets. It follows that $F^{-*}(P)$ is a vectorspace with

$$\dim(F^{-*}(P)) = \dim(P)$$

and

$$\dim(F_*(N)) = \dim(N) = \dim(P).$$

Furthermore, $F_*(N) \subset F^{-*}(P)'$ since

$$F_*(n)(F^{-*}(p)) = n \circ F_*(p \circ F^{-1}) = n(p)$$

is by definition linear for every $p \in P$ and $n \in N$. In summary, $\pi F_*(N)$ is a basis of $F^{-*}(P)'$. □

Secondly, provide means to linearly transform finite elements, by showing that the duality between degrees of freedom and nodal basis goes through a linear transformation of the degrees of freedom.

Lemma 3.2. *Let (T, P, N) be a finite element with $\dim(N) = l$ and nodal basis Ψ and let $S \in \mathbb{R}^{l\times l}$ be regular, then (T, P, NS) is a finite element with nodal basis $S^{-1}\Psi$.*

Proof. Since πN is a basis of P' and S is regular, $\pi(NS) = (\pi N)S$ is also a basis of P'. Furthermore,

$$(\pi N)S(S^{-1}\Psi) = (\pi N)SS^{-1}(\Psi) = (\pi N)(\Psi) = I.$$

□

Because $(S^{-1}\Phi) \cdot (NS) = \Phi \cdot N$, where $\Phi \cdot N := \sum_{i=1}^{\dim(N)} \phi_i n_i$ is the nodal interpolant, the two finite elements in Lemma 3.2 are interpolation equivalent.
Note, that we only considered the restrictions πN for the proof of Lemma 3.2. We therefore can generalize it to the following form:

Lemma 3.3. *Let (T, P, N) be a finite element with $\dim(N) = l$ and nodal basis Ψ and let $S \in \mathbb{R}^{l\times l}$ be regular. Consider $\tilde{N} \in \left(\mathrm{C}_{\mathrm{b}}^{\mathrm{k}}(T)'\right)^l$ such that $\pi\tilde{N} = (\pi N)S$.
Then $(T, P, \tilde{N})$ is a finite element with nodal basis $S^{-1}\Psi$.*

In contrast to before, $(T, P, \tilde{N})$ and (T, P, N) are generally not interpolation equivalent. Note, that due to the finite dimensionality of P', such a transformation matrix S exists for every choice of $\tilde{N}$ with $\pi\tilde{n} \neq 0 \in P'$ for all $\tilde{n} \in \tilde{N}$.

3.1.1 Connecting the Reference Finite Element to the Physical Finite Element

Let $(\hat{T}, \hat{P}, \hat{N})$ be a reference finite element and (T, P, N) a physical finite element, such that there exists an affine mapping $F\colon T \to \hat{T}$.

In many cases, in particular for P being a full polynomial spaces and thus all finite elements discussed in this work, we have that $F^*(\hat{P}) = P$. A famous example, where $F^*(\hat{P}) \neq P$, is the Bell element.

If $F^*(\hat{P}) = P$, then $\pi F_*(N)$ and $\pi\hat{N}$ are two bases of the same space $\hat{P}'$, which implies the existence of a matrix S such that $\pi F_*(N)S = \pi\hat{N}$. Note, that affine equivalent families form the trivial case, where we have $S = I$.

For non affine families, this allows the construction of a finite element $(\hat{T}, \hat{P}, F_*(N))$, which is defined over the reference element $\hat{T}$, and affine equivalent to the physical finite element. Its nodal basis, here denoted by Ψ^*, is given by $\Psi^* = S^{-1}\hat{\Psi}$, where $\hat{\Psi}$ is the nodal basis of the reference finite element. Most importantly, we have $\Psi^* = F^*(\Psi)$, for Ψ being the physical basis, which allows us to use Ψ^* when assembling integrals over the reference element as discussed in Section 2.4.

In lack of a better name, we will call this the *physical reference finite element*. For computational purposes, only the inverse of S is required, and by invertibility of the push-forward we obtain an equation for the *Basis Transformation Matrix* $R \coloneqq S^{-1}$, namely

$$\pi N = (\pi F_*^{-1}(\hat{N}))R. \tag{3.1}$$

Obviously, other equations for R are possible, too, but this one results in a concise notation, because for gradients it represents the chain rule, as used in the discussions below.

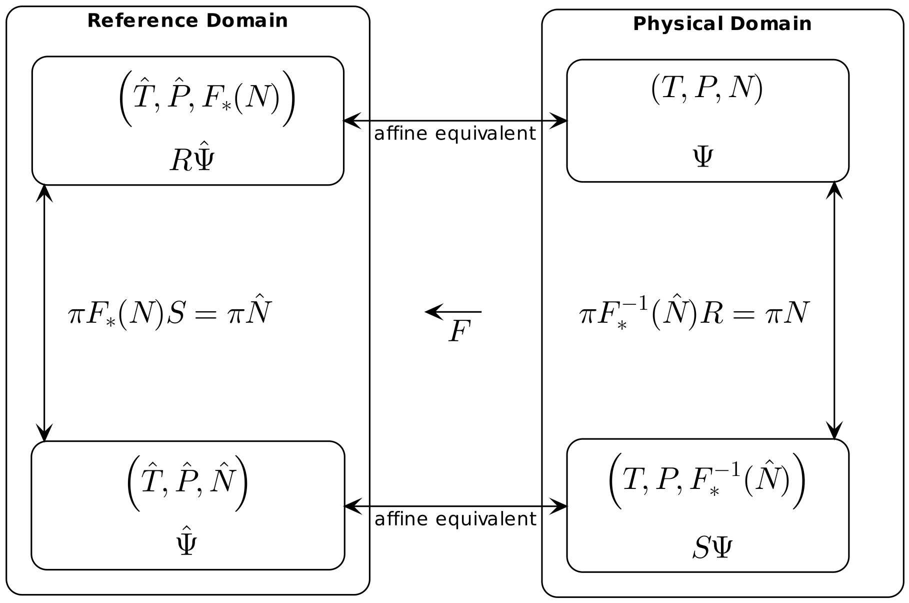

Reference Domain
Physical Domain
$\left(\hat{T}, \hat{P}, F_*(N)\right)$
$R\hat{\Psi}$
(T, P, N)
Ψ
affine equivalent
$\pi F_*(N) S = \pi \hat{N}$
F
$\pi F_*^{-1}(\hat{N}) R = \pi N$
$\left(\hat{T}, \hat{P}, \hat{N}\right)$
$\hat{\Psi}$
$\left(T, P, F_*^{-1}(\hat{N})\right)$
$S\Psi$
affine equivalent

Figure 3.1: Transformation diagram between the different finite elements and their nodal bases. The upper left finite element is the one we are interested in, but we will use the equation on the right to construct the basis transformation matrix R.

3.2 Construction of the Basis Transformation Matrix

3.2.1 Hermite Element

For every physical Hermite element (T, P, N) and an affine transformation $F\colon T \to \hat{T}$, an affine equivalent finite element defined over the reference triangle $\hat{T}$ is given by $(\hat{T}, \hat{P}, F_*(N))$ with a nodal basis $F^*\Psi = R\hat{\Psi}$, where $\hat{\Psi}$ is the nodal basis of the reference finite element $(\hat{T}, \hat{P}, \hat{N})$ and R as below. The relations between the different degrees of freedom are listed in Table 3.1a. We have

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & J_{v_0}^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & J_{v_1}^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & J_{v_2}^{-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Note that, because we can apply Lemma 3.2, interpolation equivalency is preserved, i.e., the physical reference finite element is also interpolation equivalent to the reference finite element. This is generally true for interpolation-affine families, since the relations between $F_*(N)$ and $\hat{N}$ hold in

$C_b^{k\prime}$.

$l \in \mathbb{N}$	$n \in N^l$	$F_*^{-1}(\hat{n})R_B$
1	δ_{v_i}	$\delta_{\hat{v}_i}$
2	∇_{v_i}	$\hat{\nabla}_{\hat{v}_i} J_{v_i}^{-1}$

(a) Degrees of freedom of the Hermite element. Here, $\hat{x} = F(x)$ and J is the Jacobian of F.

$l \in \mathbb{N}$	$n \in N^l$	$F_*^{-1}(\hat{n})R_B$
1	δ_{v_i}	$\delta_{\hat{v}_i}$
2	$\nabla_{m_i}^{\nu_i \tau_i}$	$\hat{\nabla}_{\hat{m}_i}^{\hat{\nu}_i \hat{\tau}_i} \hat{G}_i J_{m_i}^{-1} G_i$

(b) Completed degrees of freedom of the Morley triangle. Here, $G_i := [\nu_i \; \tau_i]^T$ and $\hat{G}_i := [\hat{\nu}_i \; \hat{\tau}_i]^T$.

Table 3.1: Global degrees of freedom of the cubic Hermite and the completed Morley triangle and their relation to the push-forward of the reference degrees of freedom. The (completed) transformation matrix R_c is block diagonal with submatrices $R_B \in \mathbb{R}^{l \times l}$.

3.2.2 Morley Triangle

Recall, that the two Morley triangles are generally not interpolation-affine equivalent, since the normal derivatives at edge midpoints are not full gradients. Accordingly, we have $\mathrm{span}(F_*(N)) \neq \mathrm{span}(\hat{N})$, but only $\mathrm{span}(\pi F_*(N)) = \mathrm{span}(\pi \hat{N})$. We can still construct a physical reference element by Lemma 3.3, but consequentially it is not interpolation equivalent to the reference finite element.

The construction of R however is more complicated. The idea is to use the finite dimensionality of P' to extend N to a completed set N_c, such that $\mathrm{span}(F_*(N_c)) = \mathrm{span}(\hat{N}_c)$ holds. By the same arguments as for Hermite element we have a matrix R_c such that $F_*^{-1}(\hat{N}_c)R_c = N_c$. The entries of R_c are given in Table 3.1b. Afterwards, we remove the additional degrees of freedom from the transformed vector.

We require the nodes $\delta_{m_i}^{\tau_i}$, where m_i is the midpoint of the ith edge and τ_i is the tangential to the ith edge. With the completed degrees of freedom at the edge midpoints containing the information of the full gradient, they can now be related to the push forwards of the reference finite element.

In this case, the simple rule

$$p'\left(\frac{a+b}{2}\right) = \frac{1}{a+b}(p(a) - p(b)) \quad \forall p \in \mathbb{P}_2\left([a,b]\right) \tag{3.2}$$

is used to create a matrix $D \in \mathbb{R}^{6\times 9}$ that transforms the original node vector

$$N = [\delta_{v_0} \, \delta_{v_1} \, \delta_{v_2} \, \delta_{m_0}^{\nu_0} \, \delta_{m_1}^{\nu_1} \, \delta_{m_2}^{\nu_2}]$$

to the extended node vector

$$N_c = [\delta_{v_0} \, \delta_{v_1} \, \delta_{v_2} \, \nabla_{m_0}^{\nu_0,\tau_0} \, \nabla_{m_1}^{\nu_1,\tau_1} \, \nabla_{m_2}^{\nu_2,\tau_2}].$$

To be precise, the rule (3.2) for a tangential derivative $\delta_{m_i}^{\tau_i}$ can be expressed in terms of degrees of freedom, that lie on the ith edge

$$\pi\delta_{m_i}^{\tau_i} = \frac{1}{l_i}(\pi\delta_{v_k} - \pi\delta_{v_l}), \tag{3.3}$$

where the ith edge has length l_i and ranges from the vertex v_k to the vertex v_l. Note, that (3.3) only holds in P', but not in $\mathrm{C}_\mathrm{b}^{\mathrm{k}\prime}$. By using this rule to transform the degrees of freedom we obtain a finite element by Lemma 3.3, that is not interpolation equivalent to the reference element.
We have

$$N_c = ND$$

with

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -l_2^{-1} & 0 & -l_3^{-1} \\ 0 & 1 & 0 & 0 & -l_1^{-1} & 0 & 0 & 0 & l_3^{-1} \\ 0 & 0 & 1 & 0 & l_1^{-1} & 0 & l_2^{-1} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

After transforming the extended node set, only the required nodes are selected by a binary matrix $E \in \{0,1\}^{\dim(N_c)\times\dim(N)}$, containing ones at the main diagonal entries that correspond to the indices of the functionals in N_c that are to remain in N.
The final transformation matrix R is given by

$$R = DR_cE. \tag{3.4}$$

3.2.3 Argyris Triangle

In principle, the approach for the Argyris element is the same as for the Morley element. A family of Argyris finite elements over a triangulation $\mathcal{T}_h$ is not interpolation-affine, again due to the presence of normal derivative degrees of freedom at the edge midpoints.
The completed set of degrees of freedom can be obtained via a rule for the tangential derivative at the edge midpoints by argument from interpolation theory, similar to (3.2).
In order to map the three unique elements of the Hessian matrix one has to apply the chain rule twice and summarize the contributions in a matrix Θ. For an affine mapping $F\colon T \to \hat{T}, (x,y) \mapsto (\hat{x},\hat{y})$,

$l \in \mathbb{N}$	$n \in N^l$	$F_*^{-1}(\hat{n})R_B$
1	δ_{v_i}	$\delta_{\hat{v}_i}$
2	∇_{v_i}	$\hat{\nabla}_{\hat{v}_i} J_{v_i}^{-1}$
3	$\triangle_{v_i}$	$\hat{\triangle}_{\hat{v}_i}\Theta$
2	$\nabla_{m_i}^{\nu_i\tau_i}$	$\hat{\nabla}_{\hat{m}_i}^{\hat{\nu}_i\hat{\tau}_i} \hat{G}_i J_{m_i}^{-1} G_i$

Table 3.2: Completed global degrees of freedom of the Argyris element and their relation to the push forward of the completed reference degrees of freedom.

one can omit the second derivatives of F and obtains

$$\Theta = \begin{pmatrix} \frac{\partial\hat{x}}{\partial x}^2 & \frac{\partial\hat{x}}{\partial x}\frac{\partial\hat{x}}{\partial y} & \frac{\partial\hat{x}}{\partial y}^2 \\ 2\frac{\partial\hat{x}}{\partial x}\frac{\partial\hat{y}}{\partial x} & \frac{\partial\hat{x}}{\partial x}\frac{\partial\hat{y}}{\partial y} + \frac{\partial\hat{x}}{\partial y}\frac{\partial\hat{y}}{\partial x} & 2\frac{\partial\hat{x}}{\partial y}\frac{\partial\hat{y}}{\partial y} \\ \frac{\partial\hat{y}}{\partial x}^2 & \frac{\partial\hat{y}}{\partial x}\frac{\partial\hat{y}}{\partial y} & \frac{\partial\hat{y}}{\partial y}^2 \end{pmatrix}. \tag{3.5}$$

For non-affine mappings, one has to incorporate the second derivatives of F together with the gradient degrees of freedom at each vertex.
The relations between the completed physical degrees of freedom and the push-forwards of reference degrees of freedom is summarized in Section 3.2.3.

3.2.4 Orientation of Normal Derivatives

For the transformations of the Morley and Argyris element above, we so far concentrated on the mapping of a single finite element onto the reference element. In this section, we discuss a compability conditions between neighboring physical finite elements.
To construct the global space in the manner of Section 2.3, one has to identify the normal derivative degrees of freedom at each edge shared by two triangles. This requires the two finite elements to agree on the orientation of their normal vectors.

A common strategy to ensure this is to introduce a global numbering of vertices in the triangulation. The tangential of an edge is then defined as the normalized vector pointing from the vertex with the lower index to the vertex with the higher index and finally the normal vector of the edge is defined as the rotated tangential. Revisiting Section 3.2.2 and Section 3.2.3, the relations given there still hold, if the normal and tangential vectors are correctly oriented throughout the family.

Another approach to this practical problem is to define ν_i as the outer normal vector of the respective triangle, t_i as its rotation, compute R as presented above and use an oriented transformation matrix $R^o = OR$. Here O is the identity matrix with diagonal entries -1 in those rows, that correspond to normal degrees of freedom that should be oriented inwards.

Note that the orientation is a non-local information, in the sense that it cannot be computed in routines that work on a single physical triangle, at least not without evaluating the global numbering. However, it can be computed and stored once, for example at the time of creation of the global basis object, and later be accessed efficiently.

3.3 Strong Enforcement of Essential Boundary Conditions

The construction of conforming spaces is the most straightforward way to enforce essential boundary conditions. Recall from Section 2.5, that in order to do so, we require $M_\Gamma \subset M$, a subset of the global degrees of freedom, that is used to encode the boundary conditions.
This set M_Γ has to be chosen such that the basis Φ_0 of the discrete space obeying the homogeneous boundary is given by $\Phi_0 = \Phi \setminus \{\, \phi_m \in \Phi : m \in M_\Gamma \,\}$, where Φ is the basis of the discrete space without boundary conditions. In the following, we consider Dirichlet boundary conditions on Γ, i.e., $\Phi_0 := \{\, \phi \in \Phi : \phi|_\Gamma = 0 \,\}$, but the discussion can be straightforwardly extended to clamped boundary conditions or boundary conditions on the union of a subset of boundary elements.

As usual, we switch from global to local perspective. For each finite element in the family whose domain T intersects with the boundary, we consider the subset $N_\Gamma \subset N_T$ of the degrees of freedom, defined by $N_\Gamma = \{\, n \in N_T : \exists m \in M_\Gamma \text{ with } m|_{\mathrm{C}^k_\mathrm{b}(T)} = n \,\}$. For implementational purposes, the goal is to construct N_Γ for each element, such that we can construct M_Γ as their union. Note, that for finite elements of Lagrange type, N_Γ contains all degrees of freedom located on $\Gamma \cap T$. However, generally and particularly for finite elements of Hermite type, this is not the case.

Furthermore, for finite elements with derivative degrees of freedom, the suitable choice for N_Γ is not always a subset of N_T. Consider for example in the standard definition of the Hermite element, where the derivatives are evaluated in the directions of the global coordinate axes. Additionally assume, that T and Γ are such, that v_1 and v_2 are the endpoints of an edge $e \in T \cap \Gamma$. Then for $i \in \{\, 1, 2 \,\}$, the degree of freedom $\delta^w_{v_i}$ is only an element of $P(T \cap \Gamma)'$, if w is tangential to e at v_i. Consequentially, the set $\{\, \delta_{v_1}, \delta^w_{v_1}, \delta_{v_2}, \delta^w_{v_2} \,\}$ does only specify the values of a cubic polynomial along e and thus would be a suitable choice for N_Γ, if w is tangential to e. Thus, if neither of the coordinate directions x and y is tangential to e, there is no suitable choice for N_Γ. One could use the whole gradient at v_1 and v_2, in order to implicitly set $\delta^{\tau_e}_{v_1}$ and $\delta^{\tau_e}_{v_2}$, but if either v_1 or v_2 is not a corner, this leads to a discrete boundary condition that the true solution does not necessarily fulfill. In other words, the interpolation properties on the elements intersecting the boundary are lost and decrease (or even total loss) of convergence is to be expected. By Céa's lemma, this translates directly into the convergence of the finite element method.

The consequence of the discussion above is that one can only construct Dirichlet condition conforming subspaces using standard Hermite elements by the means developed so far, if the boundary is aligned to the coordinate axes. Note that this is trivially given for one-dimensional domains. In the two-dimensional case, one can use the theory established in the previous sections to define finite elements with *grid aligned* derivative degrees of freedom at the boundary. For every vertex v at the boundary with tangentials τ_1 and τ_2 and normal ν, instead of the gradient ∇_v the grid aligned Hermite element contains the degrees of freedom $\nabla_v^{\tau_1,\tau_2}$, if τ_1 and τ_2 are linearly independent, or $\nabla_v^{\tau_1,\nu}$, if v is a vertex on the interior of a straight boundary segment as in the example above. The approach can be straightforwardly extended to the Argyris element by replacing $\triangle_v$ with $\triangle_v^{\tau_1,\tau_2}$ or $\triangle_v^{\tau_1,\nu}$.

The degrees of freedom N_t of the axis aligned Hermite element can simply be constructed from the usual Hermite element by

$$N_t = N \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (v_1\, w_1) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (v_2\, w_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (v_3\, w_3) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.6}$$

where $(v_i\, w_i)$ is the matrix with the two directions at the ith vertex as columns. In three dimensions, the situation is less clear, since a vertex at the boundary has generally arbitrarily many tangentials. In principle, a similar approach could be implemented, that iterates over the tangentials, adds them to the direction set if linear independent and afterwards fills the direction set with a normal vector if needed.

Using this tangential and normal degrees of freedom, the set N_Γ of degrees of freedom used to construct the Dirichlet condition conforming subspace contains all degrees of freedom located on the boundary, which are either function evaluations or tangential derivatives. The concrete choices of N_Γ for the Hermite and Argyris element are listed in Table 3.3. Not contained in this table are fourth order problems with simply supported boundary conditions, since there we have the same situation as for second order problems with Dirichlet conditions. Note, that with a more precise distinction of degrees of freedom, one can also give a set $N_{\Gamma,T} \subset N_\Gamma$, a subset that only contains those degrees of freedom needed to incorporate the respective boundary condition locally on each element T. The global set can still be constructed by $M_\Gamma = \bigcup_{T\in\mathcal{T}_h} N_{\Gamma,T}$. This allows one to include different boundary conditions on different parts of Γ, with one exception. When discretizing a fourth order problem with the Argyris element, a switch from clamped to free boundary conditions might happen at a non-rectangular corner. In this case we would require the degrees of freedom at this corner to be directed into the tangential τ_c and normal direction ν_c of the clamped part, such that we can set $\delta_v^{\tau_c,\nu_c}$ to zero, without implying unnecessary conditions on the free part. However, with the generic choice of directions described above the degrees of freedom would be directed into the

Element	Condition	straight boundary	corner
Hermite	Dirichlet	$\delta_v\ \delta_v^{\tau}$	$\delta_v\ \delta_v^{\tau_1}\ \delta_v^{\tau_2}$
Argyris	Dirichlet	$\delta_v\ \delta_v^{\tau}\ \delta_v^{\tau\tau}$	$\delta_v\ \delta_v^{\tau_1}\ \delta_v^{\tau_2}\ \delta_v^{\tau_1\tau_1}\ \delta_v^{\tau_2\tau_2}$
Argyris	Clamped	$\delta_v\ \delta_v^{\tau}\ \delta_v^{\tau\tau}\ \delta_v^{\nu}\ \delta_v^{\nu\tau}$	N_v

Table 3.3: Elements of N_Γ located at a vertex v for Dirichlet or clamped boundary conditions. N_v is the set of all degrees of freedom located at a vertex v.

direction of the tangentials to both boundary segments meeting at the corner, and thus we would have to set the whole Hessian to zero in order to set the second normal derivative. Consequentially, for problems with such a switch, one needs to define a basis with directional derivatives that depart from the generic choice in this corner.

Note, that the choice of the vector (v, w) for each vertex is again a non-local property, since is only can be computed by the information obtained from the two triangle with boundary edges, that intersect at the respective vertex. Together with the orientation of the normal vectors, this motivates the implementation of a structure holding these non-local information for each element, as will be proposed in Section 4.2.1.

Remark 3.1 (Scope of the tangential approach). The presented approach has some drawbacks that should be mentioned.

As a first, it does require the boundary data β, that encodes the boundary condition with some differential operator B in the form $B(u) = \beta$ on Γ, to be defined on a neighborhood of Γ, such that all derivatives can be evaluated.

Secondly, clamped boundary conditions are often given in a form that separates the condition on function values from the condition on the normal derivative. However, the intersection of the two sets of degrees of freedom needed to incorporate those conditions is not necessarily empty. Take for example a problem with clamped boundary conditions in the form of

$$u = f \text{ on } \Gamma \tag{3.7}$$

$$\partial_\nu u = g \text{ on } \Gamma, \tag{3.8}$$

that is to be discretized with the Argyris element. At a corner v_0 of the domain, enforcing (3.7) requires setting $\delta_{v_0}^{\tau_0}$ and $\delta_{v_0}^{\tau_1}$, but so does enforcing (3.8). In order to construct the boundary interpolant u_{hc} as presented in Section 2.5, we have to evaluate $m(h)$ for all $m \in M_\Gamma$ for a function h satisfying

$$h = f, \quad \partial_\nu h = g \text{ on } \Gamma.$$

Clearly such a function h, that encodes the data for all boundary conditions, is not trivially given for many problem settings. However, the method is applicable to all problem settings where the boundary data is defined by the traces of some function h. Also, one might argue, that problems, where such h does not exist, are not well-defined in a strong sense.

Finally, since the grid aligned nodal basis involves the inverse of $(v_i w_i)$, numerical issues can arise, when the two tangential vectors become close to parallel. Hence, the grid aligned finite elements should be used carefully, when approximating a boundary with very small but nonzero curvature.

4 Implementation

4.1 Dune

DUNE, the *Distributed and Unified Numerics Environment* [Blatt et al., 2016], is a modular C++ framework aiming at flexible but easy implementation of finite element, as well as finite volume and finite difference Methods. This chapter mainly aims at the description of the implementation of the three types of finite elements discussed in the previous chapters. To do so, first some of the most important interfaces of DUNE are introduced. However, a complete description of DUNE and its capabilities is out of the scope of this thesis. For a thorough introduction the reader is referred to [Sander, 2020].

Throughout DUNE, little to no inheritance is used. Instead, DUNE uses a template based Duck typing approach. This means, that interfaces are only defined as concepts, i.e., a description of method signatures and exported types a class has to provide in order to implement the interface. However, it is not always checked that a class fulfills the whole interface, or that those methods work as expected. The advantage of this duck typing approach is that the compiler knows the full types at compile time, and can thus apply his full range of optimizations. At the same time, this leads to generic routines that can be used by different types implementing the same interface, like different finite elements implementing the `LocalFiniteElement` interface, detailed out below. Before we can turn to the finite elements implemented in the context of this work, we have to discuss some of the most important interfaces in DUNE.

4.1.1 The `dune-core` Modules

DUNE is structured in modules. The most important group of modules is called `dune-core` modules and covers elementary parts of the aforementioned methods.

In particular, `dune-grid` [Bastian et al., 2008a, Bastian et al., 2008b] provides the `Grid` interface and some implementations of it. Additional modules provide more general types of grids or wrap other grid managers. The capability of DUNE to provide a variety of different grid managers through a common interface is one of its strengths. After creation of the `Grid` object, most of the code will only get in contact with the `GridView` object. This is a immutable view on the grid, allowing traversal over entities of all codimensions, that is triangles, edges and vertices in the case considered here. In other words, it models *a* triangulation $\mathcal{T}_h$, whereas the `Grid` models a family of triangulations by offering refinement methods.

Structurally separated from the `Grid` and its elements is the interface to the reference element and the mapping from the reference element onto the physical elements. All of this is contained in the module `dune-geometry`, which also provides quadrature rules on the reference element.

As the arguably most important `dune-core` module in the context of this work, the module `dune-localfunctions` provides the `LocalFiniteElement` interface, which mirrors the definition of a reference finite element quite closely. It provides the means to evaluate the nodal basis at a *local* point, that is, a point in the reference domain, and offers access to `LocalInterpolation` objects, which model the nodal interpolation operator of a finite element. In Figure 4.1 the structure of `LocalFiniteElement` is presented. Along with the general interface, the module provides a variety of classes fulfilling this interface, ranging from the classical Lagrange finite elements to $\mathrm{H}_{\mathrm{div}}$ conforming elements. So far however, it contains no finite element of class C^1.

The module `dune-istl` [Blatt and Bastian, 2007] offers an abstract linear algebra environment, implementing solver and preconditioner routines to be used in sequential or distributed settings.

Finally, the `dune-core` modules are completed by a module named `dune-common`, which offers a collection of tools used in all other modules like the basic matrix and vector types, but also the build system.

The other modules mostly fall into one of the following groups:

Grid modules As mentioned, many of the concrete implementation of the `Grid` interface have their own module. Some implement the interface directly, some serve as adapters to third party grid managers, and some are wrappers around other grid implementations.

Discretization modules These modules provide implementations of concrete discretization schemes and methods. They vary in the type of methods implemented, degree of abstraction and other aspects like the degree of parallelization.

Extension modules While some of them extend modules of the above listed groups, others provide additional tools of varying complexity. Most notably, the module `dune-typetree` provides tree structures, that can be evaluated at compile time and the `dune-functions` module provides a first layer of abstraction, by formalizing global functions and finite element spaces. Due to its significance, it will be discussed separately in the following chapter.

4.1.2 The `dune-functions` Module

The `dune-functions` module provides interfaces that connect localized (finite element) functions and global functions defined over a grid. It therefore depends on `dune-localfunctions`

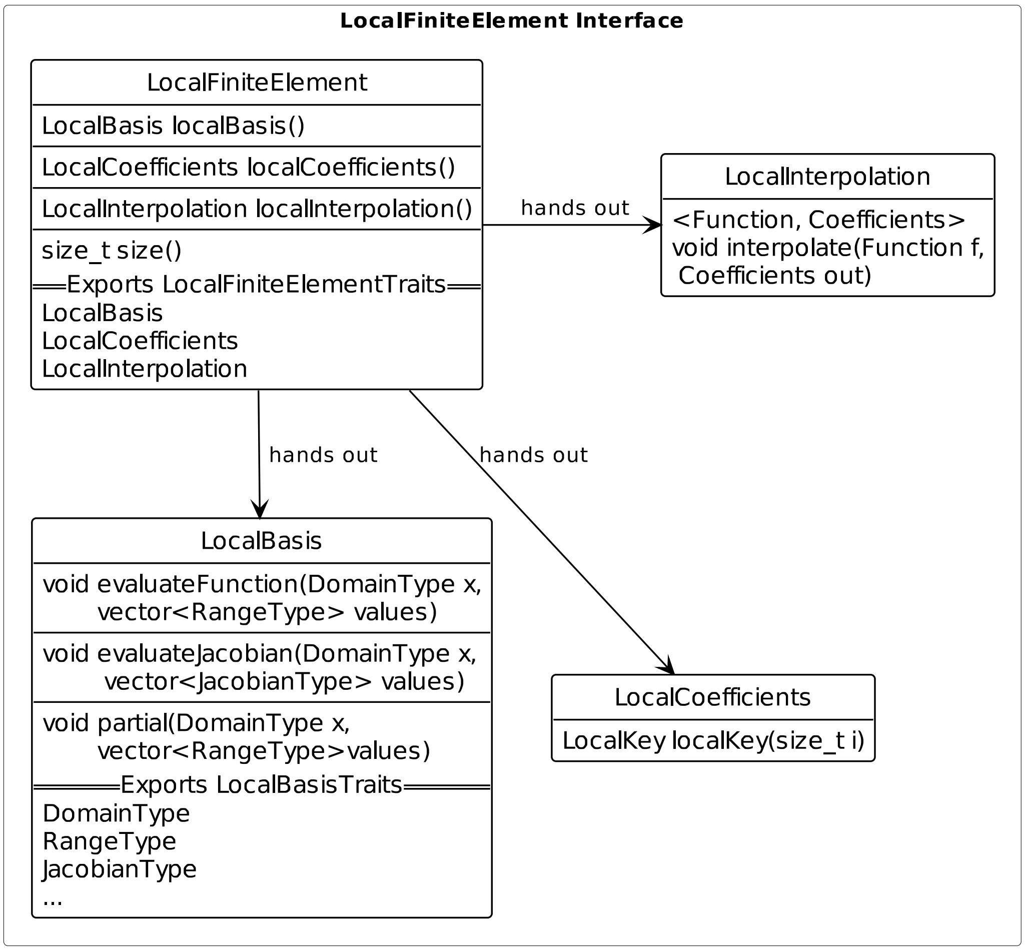

LocalFiniteElement Interface
LocalFiniteElement
LocalBasis localBasis()
LocalCoefficients localCoefficients()
LocalInterpolation localInterpolation()
size_t size()
Exports LocalFiniteElementTraits
LocalBasis
LocalCoefficients
LocalInterpolation
hands out
LocalInterpolation
<Function, Coefficients>
void interpolate(Function f, Coefficients out)
hands out
hands out
LocalBasis
void evaluateFunction(DomainType x, vector<RangeType> values)
void evaluateJacobian(DomainType x, vector<JacobianType> values)
void partial(DomainType x, vector<RangeType>values)
Exports LocalBasisTraits
DomainType
RangeType
JacobianType
...
LocalCoefficients
LocalKey localKey(size_t i)

Figure 4.1: Most important methods of the `LocalFiniteElement` interface. The concrete argument and return types for the methods can be statically obtained from the exported `Traits` types.

and `dune-grid` and offers an implementational model of discrete function spaces. Together with the `dune-istl` module, it completes the basic toolbox for finite element routines.

Interface for Functions As the name indicates, `dune-functions` provides an interface for functions. It distinguishes between *global* functions, to be evaluated in global (physical) coordinates, and *local* functions, to be evaluated on the reference element. This distinction sets the environment for a consequential application of the reference paradigm, as introduced in Section 2.4. This section briefly summarizes the interface, for more details, see [Engwer et al., 2017].

`DifferentiableFunction` The main interface for functions (either global or local) is the class `DifferentiableFunction<Range(Domain),DerivativeTraits>`. It is a wrapper class, that implements a type erasure, similar to `std::function`, but adapted to the mathematical setting. As indicated by the name, this interface models a differentiable function f. In particular, it provides the evaluation operator

```
Range operator()(Domain const & x) const;
```

which accepts an argument x of type `Domain` and returns the result of $f(x)$ as an object of type `Range`. Those two types form the *signature* of the function and are given, together with a type `DerivativeTraits` describing the hierarchy of types returned by the derivatives of f, as template parameters. The derivative of f is obtained by the free function

```
auto derivative(DifferentiableFunction<...> const & f);
```

Note that there is no automatic differentiation, instead the user has to provide the derivatives upon creation of the `DifferentiableFunction`.

`LocalFunction` A function defined on the reference element is modeled by the interface `LocalFunction`[1]. Mathematically, for a given function f and a triangulation $\mathcal{T}_h$, it models the set of pull-backs $\{ F^T_*(f)\}_{T\in\mathcal{T}_h}$, where $F^T : \hat{T} \to T$ is the invertible mapping from the reference element to the physical element. It fulfills the `DifferentiableFunction` interface, but additionally provides a method

```
bind(Element const& T);
```

which selects the concrete pullback. Calling the evaluation method before the `bind` method is undefined behaviour.

Mathematically, the derivative $DF_*(f)$ of a pull-back does not equal the pull-back of the derivative $F_*(Df)$. However, since $DF_*(f)$ is rarely needed, the `dune-functions` module opted for a mathematical inconsistency here, and so

```
derivative(LocalFunction<...> const & f);
```

returns $F_*(Df)$.

Interface for Discrete Spaces The `dune-functions` interfaces of finite spaces model the structure laid out in Section 2.3 quite closely. The distinction between the global perspective and the local perspective is explicitly modeled by separation of interfaces. At the top of the hierarchy is the `GlobalBasis` interface, which models the global basis Φ, obtained by combining local basis functions across elements. Switching from global to local perspective is done by calling `globalBasis.localView()`, which returns an object implementing the `LocalView` interface. It provides all the methods needed to assemble the integrals over the reference domain $\hat{T}$. For more

[1]Strictly, the object described here is a `GridViewFunction`, which is a concept refining the more general concept of a `LocalFunction` by only accepting the elements of a `GridView`. A `LocalFunction` would accept all elements of suitable type, independent of whether the particular element is contained in a particular triangulation.

details, for example on indexing strategies and more, see [Engwer et al., 2018]. The structure of the interfaces explained below is sketched in Figure 4.2.

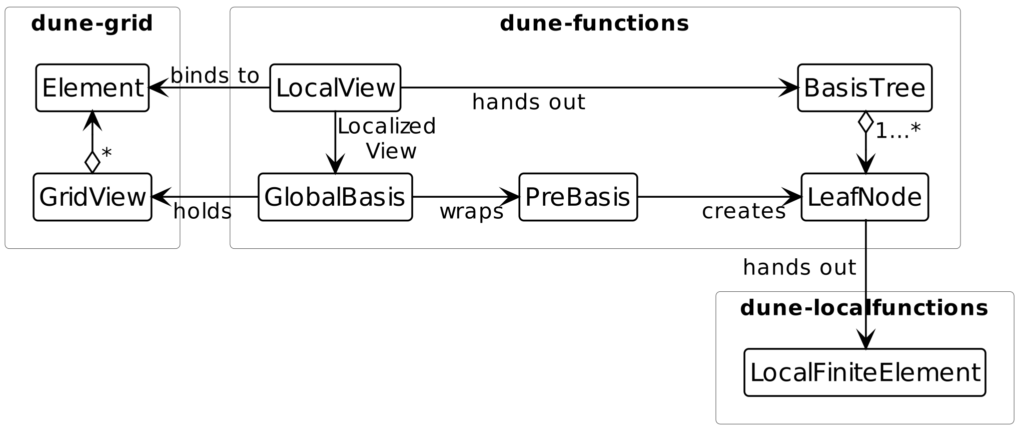


Figure 4.2: Broad structure of the `dune-functions` interfaces for finite element spaces. An asterisk denotes variable cardinality of an aggregation.

`GlobalBasis` At the top of the hierarchy of interfaces in `dune-functions` is the `Global-Basis` interface. A `GlobalBasis` models a discrete function space basis defined over a `GridView`. It hands out a localized view on the basis, the `LocalView`, and manages the construction of global indices. While one can use specific implementations of the `Global-Basis` interface, DUNE provides a default implementation that relies on a `PreBasis`. In the language of design patterns, the `DefaultGlobalBasis` offers a number of template methods, while the `PreBasis` implements the hooks called by those methods.

`PreBasis` Global properties of a finite element space are essentially encoded in the implementations of the `PreBasis` interface. It provides methods to access the global information needed to work with the values obtained from the finite elements. In particular, it provides a `size()` method returning the number of global basis functions as well as the method `in-dices(...)` used to assign a global numbering to the local basis functions. In other words, when the `PreBasis` implementation assigns the same global index to two or more local basis functions, it identifies their degrees of freedom in the manner of Section 2.3. Different implementations of the interface can hence implement different finite element spaces using the same `LocalFiniteElement` implementations, for example discontinuous and continuous Lagrange finite element spaces.

`LocalView` Typical assembling routines traverse over the elements of the triangulation and assemble the subintegral on each element. The `LocalView` interface models the restriction of a global to an element and therefore everything needed for the assembling of the integral over this element.

In particular, it offers access to the basis tree, explained below, by the method `tree()` and provides with the method `index(Sizetype i)` the mapping from the local index of a reference basisfunction to a global (multi-) index of the corresponding global basis function needed to specify the correct entry in the linear equation system.

Similar to the `LocalFunction` interface, the `LocalView` has to be bound to an element `e` of the `GridView` via the method `bind(Element const& e)` in order to be usable. This method start a chain of calls to various variations of `bind(...)` methods in potentially all objects, to which the `LocalView` indirectly offers access. This includes all `Nodes` in the `Basistree`, and the corresponding `LocalFiniteElement` objects they hold. This call chain, called *binding*, is a central concept in the `dune-functions` interface. It allows the implementations to collect expensive computations in a method that should only be called once per element when iterating over a triangulation.

`Basistree` The basistree [2] returned by the method `localView.tree()` models a hierarchy of finite elements that form the (physical) reference finite element. For a scalarvalued basis, for example to be used for discretizing the Poisson equation, this tree only contains one node. For vectorvalued finite element spaces, mixed methods and so on, the tree contains more than one basis. `LeafNodes` are (mostly scalar) finite elements while inner nodes can be either `power` nodes, containing multiple child nodes of the same type, or `composite` nodes, containing child nodes of different types.
Take for example the Taylor-Hood element of degree k, used for discretization of the Stokes equations. It contains a scalar Lagrange basis of degree k for the pressure and a vectorvalued Lagrange basis of degree $k+1$ for the velocity. The corresponding tree would be a `composite` node of a Lagrange node of degree k and a `power` node of a Lagrange node of degree $k+1$.

`LeafNode` `LeafNodes` of the tree offer the `finiteElement()` method which returns an object implementing the `LocalFiniteElement` interface. Additionally, they offer a method `localIndex(Sizetype i)` with returns the local index of their ith basis function in the root tree. The returned value can then be mapped to the global index by `localView.index(Sizetype j)`.

Note, that the `LeafNode` method `finiteElement()` does not have to return a object of a class defined in `dune-localfunctions` but rather an object of a class that fulfills the interface.

[2]To precise, the method returns the root of said basistree

It is hence a suitable place to return transformed finite elements. An example is the class `GlobalValuedLocalFiniteElement` used to implement finite elements which utilize a range space transformation, like the Piola transformation in case of the Nédélec element and others. While the class implements the `LocalFiniteElement` interface, it is itself a wrapper around a corresponding class from `dune-localfunctions`. Without going into the details here, it is to be mentioned that this class served as a orientation for the classes implemented for the finite elements described in Section 3.2. It is possible to implement those finite elements using `GlobalValuedLocalFiniteElement` as well, but the class lacks the possibility to cache the transformation for multiple evaluations on the same element, which is considered necessary given the expensive construction of the transformation matrices.

4.2 Implemented Elements

We present a module named `dune-c1elements`[3], which contains the implementations of the finite elements we discussed in previous chapters. It is an extension module for `dune-functions`. As such, it depends on `dune-functions` and its dependencies, namely the `dune-core` modules and `dune-typetree`.
As pointed out, the appropriate point to implement transformed finite elements is the return type of the `finiteElement()` method. For finite elements using linear transformations of the reference nodal basis, this will be an object of the class `LinearTransformedLocalFiniteElement`.

4.2.1 The `LinearTranformedLocalFiniteElement` Class

As the name indicates, this class models the linear transformation of a finite element on the reference element, as discussed in Chapter 3. It is reusable in the sense, that all implementations use variations of it by supplying different template parameters. Therefore, it is a template class in a C++ sense, but also in the sense of design patterns. The class declaration reads

```
template <typename LinearTransformator,
          typename LocalFiniteElement,
          typename Element>
class LinearTransformedLocalFiniteElement;
```

The template parameters mirror the general approach of the class. They encode any information on the actual finite elements and their transformation, while the `LinearTransformedLocalFiniteElement` class wraps them and provides the `LocalFiniteElement` interface to be used by higher level code like assembling routines.
The first template parameter is a class `LinearTransformator`, which, in the language of Sec-

[3]Available at https://gitlab.dune-project.org/maik.porrmann/dune-c1elements

tion 3.1, models the transformation from the reference finite element to the physical reference finite element. This transformation is objectified and the corresponding object is included in the binding process. This allows the aforementioned caching of the transformation matrix, which is filled upon the call to `bind(...)` and reused for multiple evaluation calls on the same element.
The second template argument is the type of the reference finite element. The `LinearTransformedLocalFiniteElement` class holds a pointer to the reference finite element and access its methods to evaluate the reference nodal basis and to forward the `LocalKey` objects for each degree of freedom.
Lastly, the classes are templated by the type of the element that they can be bound to[4].

Evaluation of the Nodal Basis The `LocalBasis` interface from `dune-localfunctions` does not grant access to single basis functions of a finite element, but only allows evaluation of the whole basis altogether. This means that the implementation of the evaluation of the transformed basis simply is a two-step procedure. First, the reference basis $\hat{\Psi}$ is evaluated at a point x, then the return values are computed by a matrix vector multiplication $R\hat{\Psi}(x)$. Note that since the `dune-istl` interface of a matrix vector multiplication does not mandate the vector to be a vector of scalars, but only to be a vector of objects, which themselves implement vector space operations, this approach works for the evaluation of the Jacobians and Hessians of the basis functions identically. Accordingly, the `LinearTransformator` class does not implement different methods for those cases, but simply one method that accepts vectors of vectors or vectors of matrices as well, which mirrors the mathematical property of a linear transformation of basis functions.

Non-local Information As mentioned at various points in Chapter 3, the implemented elements require a certain amount of non-local information, that is information which is not obtainable from one physical element or from the reference element. This includes

- the orientation of normal vectors for each edge,
- the directions of derivatives used as degrees of freedom,
- the set $N_{\Gamma,T}$ of degrees of freedom to be used when incorporating essential boundary conditions on an element $T \in \mathcal{T}_h$.

This information is collected upon the creation of the `PreBasis`, saved in a dedicated structure, called `ElementInformation`, and passed down from the `LeafNode` to the `LinearTransformator` and the `LocalFiniteElement` objects during the binding process. Note that there is not direct way to access this information, except for the information, whether a specific degree of freedom is contained in $N_{\Gamma,T}$. This access will be discussed in Section 4.3.

[4]This is actually the concrete type of the implementation and differs for different grid managers

Interpolation Traditionally in DUNE, the global interpolation routine uses the `LocalInterpolation` object, which models the nodal interpolation operator of the reference finite element and is obtained from the `LocalFiniteElement`. In the linearly transformed case however, this approach only mirrors the mathematical framework correctly, if the physical reference element and the reference finite element are interpolation equivalent. In our case, this is only true for the Hermite element.
To account for the lack of interpolation equivalency properly in the class structure and also because this allows a more efficient implementation, the `LinearTransformator` class can export a `GlobalInterpolation` class, which is included in the binding process and is similar to the `LocalInterpolation` interface, meaning it implements a method

```
template <class F, class C>
interpolate( F const & f, std::vector<C> & out);
```

which fills the coefficient vector `out` with $(n(f))_{n\in F_*(N)}$ for a `LocalFunction f`. However, since the degrees of freedom contain the evaluation of derivatives, the object `f` to be interpolated has to offer the amount of derivatives needed by the respective finite elements nodal interpolation operator. This is a violation of the `LocalInterpolation` interface, which mandates the `interpolate(f,out)` method to work if `f` only provides the evaluation operator.

The structure of the `LinearTransformedLocalFiniteElement` class and the interplay with the `LinearTransformator` interface, as discussed below, is sketched in Figure 4.3.

4.2.2 The `LinearTransformator` Interface

In contrast to the template class `LinearTranformedLocalFiniteElement`, the `LinearTransformator` is an interface, which is implemented by a class for every finite element we considered. It provides the methods and classes to be used by the template class `LinearTranformedLocalFiniteElement`. Most parts of the interface have already been mentioned. For completeness, they are listed here again.

- It sets up the transformation upon a call to `bind(Element const& e, ElementInformation const& eInfo)`.
- It applies the linear transformation on a vector of scalars, vectors or matrices via the `apply(vector const& in, vector& out)` method.
- It exports the class `ElementInformation`.
- It exports the class `GlobalInterpolation`.

4.2.3 Concrete Implementations

With interfaces explained in the previous chapters, the concrete implementation for each of the transformed finite elements is given by the following classes, here named by the interface they implement. The concrete classes in the module are prefixed by the name of the finite element.

- The classes to implement the `LocalFiniteElement` interface, modeling the reference finite element. This amounts to a class `LocalBasis`, which allows the evaluation of nodal basis functions and their derivatives, a class `LocalInterpolation` which models the reference nodal interpolation operator, a class `LocalCoefficients`, which provides the `LocalKey` for each degree of freedom, and lastly a class `LocalFiniteElement` offering access to the objects above.

- A class implementing the `PreBasis` interface. Additionally to the aforementioned encoding of global properties, this class also collects non-local information upon creation and holds a map which assigns each element of the `GridView` its `ElementInformation` object. For the finite elements implemented so far, this map is objectified by a class `ElementInfor-mationMap`, which is however not part of the public interface.

- The `Node` class mainly connects the different classes. Most importantly, it defines the correct return type of the `finiteElement()` method. Furthermore, during binding, the `Node` adds the `ElementInformation` object for the respective element as well as the reference finite element to the argument list when calling the `bind(...)` methods of the transformed finite element.

- The largest interface to fulfill is the `LinearTransformator`. Almost all the other classes are parametrized with this class, mostly to get the types it exports, but also, because the transformation is at the center of the mathematical theory and turns the other classes into a working implementation of a transformed finite element space.

- As a very last point to note, all finite elements offer a factory method, like the method `her-mite<dim>()` for the Hermite element, which returns a `PreBasisFactory`. This factory object can be used with a `GridView` object to create a global basis, for example:

  ```
  auto preBasisFactory = hermite<dim>();
  auto globalBasis = makeBasis(preBasisFactory,gridView);
  ```

  The object `globalBasis` offers the `GlobalBasis` interface and models a Hermite finite element space in `dim` dimensions.

Summarizing the implemented interfaces, we state, that the new finite elements can be used straight-forward in any assembling routine and interpolation routine, provided the former avoids caching and the latter hands down a function object that is differentiable appropriately many times. A proposal implementation of such a method in contained in the module. Discretization modules, that want to include our module, would have to adapt their interpolation routine as well as their interface for Dirichlet or generally essential boundary conditions, as discussed below.

4.3 Essential Boundary Conditions

The design of an interface that allows a simple incorporation of different classes of essential boundary conditions might be the most critical part of this implementation. This is due to several reasons. First of all, the finite elements as well as the `LinearTransformedLocalFiniteElement` class form an extension to the `dune-functions` module. In contrast to that, the interface for boundary conditions generally is part of the more high level discretization modules. Consequentially, there is no official interface for boundary conditions throughout the DUNE world. Furthermore, the exemplary implementation given in [Sander, 2020], which is in fact used with some modification by some discretization modules, uses all degrees of freedom on the boundary to incorporate Dirichlet conditions. As discussed in Section 3.3, this is not possible for the finite elements of Hermite type.

In order to allow the construction of boundary condition conforming spaces, the implementations provide methods to answer the question, whether a specific degree of freedom is contained in the set $N_{\Gamma,T}$ for Dirichtlet or clamped boundary conditions on a triangle T. Recall that $N_{\Gamma,T}$ does not contain degrees of freedom, that are needed to incorporate the boundary condition on a neighboring element. Therefore, including different boundary conditions on different parts of the boundary can be implemented straightforwardly, with the mentioned exception of a switch from clamped to free conditions at a non-rectangular vertex.
Note, that these methods are not part of the official DUNE interface, and so there was no a priori suitable place to implement them. As of now, they are part of the class `LinearTransformed-LocalCoefficients`, an object of which is obtainable from the `LinearTransformedLo-calFiniteElement` class. It implements the `LocalCoefficients` interface, which usually only provides the `LocalKey` objects for each degree of freedom. The `LinearTransformed-LocalCoefficients` class provides two additional methods, namely

```
bool isDirichlet(size_t i) const;
bool isClamped(size_t i) const;
```

which return `true` iff the `i`th degree of freedom is contained in $N_{\Gamma,T}$ for the respective kind of boundary condition. Internally, the class accesses the `ElementInformation` object it is bound to, which can implement a corresponding method, whose result is forwarded. If the `ElementIn-formation` class does not implement such a method, the methods above always return `true`.

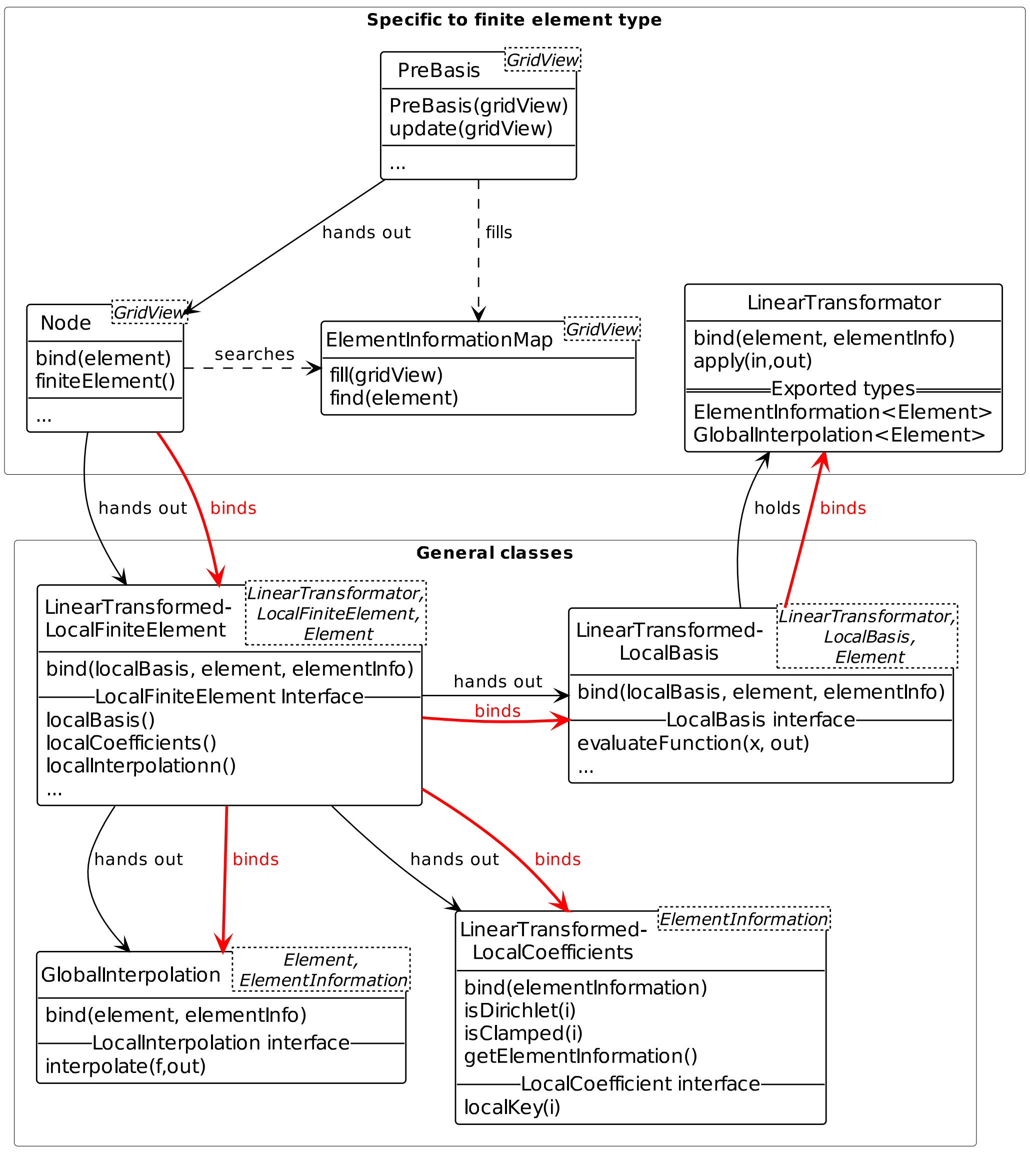


Figure 4.3: Class diagram for linear transformed finite elements. The upper part is specific for each element, i.e., there is a class `HermitePreBasis`, a class `ArgyrisPreBasis`, and so on. The classes in the lower part are reusable. Template parameters are written in dashed boxes of each class. Note, that this diagram omits a great number of methods, all type information for methods as well as some template parameters like those for Range and Domain.

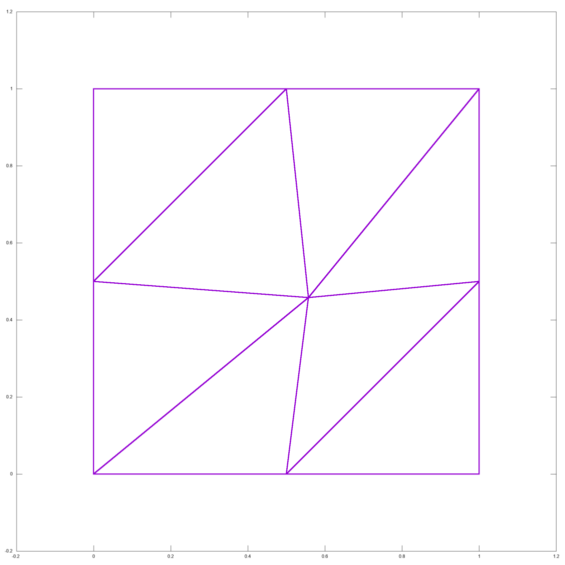
5 Numerical Experiments

This section contains a number of numerical experiments of increasing complexity, aiming to demonstrate the capabilities of the implemented finite elements. At first, the properties of the global interpolation operators are verified, that is, the global interpolation should be (numerically) exact for polynomials up to the respective polynomial degree k of the finite element, and show an interpolation error in terms of the L^2-norm of order h^{k+1} for higher degrees. Secondly, the implemented finite elements are used to discretize a number of suitable boundary value problems, ranging from a Reaction-Diffusion equation with Neumann boundary conditions as the simplest case to the clamped plate problem as a classical example of a fourth order problem.

Throughout this chapter, the problems are solved on a series of uniform red refinements of an initial grid. Whenever easily possible, a fully perturbed grid, that is a grid obtained from a uniform square grid by a perturbation of all vertex positions, is used. In some cases, manufacturing a solution is easier on a rectangular boundary, in those cases only the position of the inner vertex is perturbed. Both grids are shown in Figure 5.1.

5.1 Interpolation

As a first step, we consider the nodal interpolant as given in Definition 2.5. The theoretical interpolation estimates for interpolation involving derivative degrees of freedom are of the same quality as those for Lagrange interpolation, as detailed out in [Ciarlet and Raviart, 1972], i.e., so long as we have a unisolvent set of functionals on (a superspace of) a polynomial space of degree k, the L^2 interpolation error for suitable functions is of order h^{k+1}. However, since the global interpolation operator for elements of Hermite type maps into a strict subspace of the range of the interpolation operator of Lagrange finite elements, the error is expected to be higher in absolute terms. Furthermore, for both types of interpolation, the interpolation trivially fulfills $v - \mathcal{I}_h v \equiv 0$ for all $v \in V_h$ and since the global polynomial space $\mathbb{P}_l(\Omega)$ is a subspace of V_h for $l \leq k$, the interpolation is exact in those cases. The interpolation errors for the different finite elements are plotted in Figure 5.2 for the one-dimensional case and in Figure 5.3 for two dimensions, each accompanied by the error for Lagrange finite elements of the same degree. They show the expected behavior described above, i.e., (numerical) exactness for polynomials of less or equal degree, respectively, as well as the same order of convergence for the general case, albeit generally a higher error, as the Lagrange interpolation


(a) Square grid

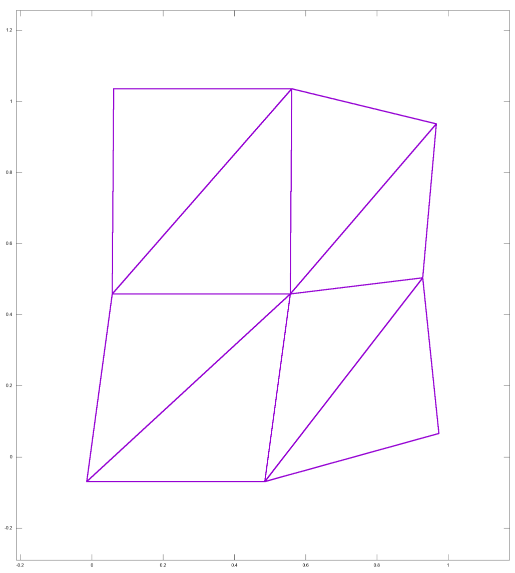

(b) Perturbed grid

Figure 5.1: Grids used for numerical experiments.

operators.

5.2 Second Order Problems

As the most simple example, we consider a second order problem with Neumann conditions. This avoids any difficulties in the construction of boundary condition conforming subspaces, since we have natural boundary conditions. We compare the error $\|u - u_h\|_{1,2,\Omega}$ obtained by a standard finite element routine using Hermite elements, Argyris elements and Lagrange elements of degree 3 and 5.

Example 5.1 (Reaction Diffusion Equation with Neumann Conditions). We manufacture a strong solution $u(x,y) = x^n + y^n$ for some $n \in \mathbb{N}$ with $n > 1$. The right hand side of the strong form is chosen as

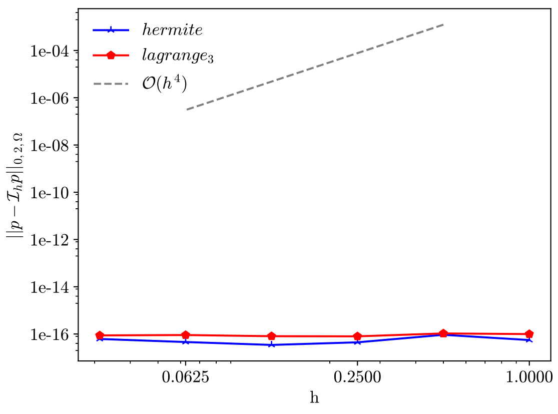
$$f(x,y) = -(n-1)n\left(x^{n-2} + y^{n-2}\right) + x^n + y^n,$$

such that u solves the weak problem

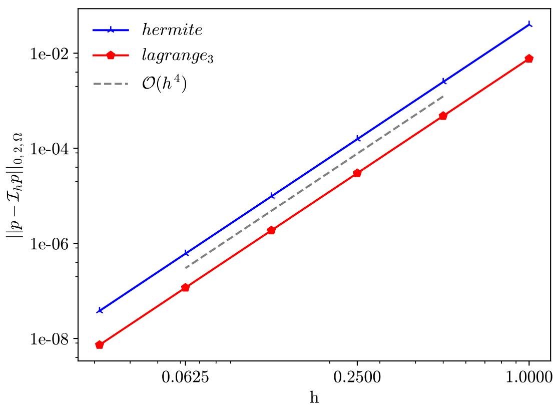
$$\int_\Omega \nabla u \nabla v + \int_\Omega uv = \int_\Omega fv + \int_\Gamma v\partial_\nu u \quad \forall\, v \in \mathrm{H}^1(\Omega), \tag{5.1}$$

for some domain Ω with boundary Γ.

The errornorms for Example 5.1 solved on the fully perturbed grid are depicted in Figure 5.4.



(a) $deg(p) = 3$



(b) $deg(p) = 4$

Figure 5.2: Interpolation error $\|p - \mathcal{I}_h p\|_{\mathrm{L}^2}$ for one-dimensional Hermite element and Lagrange element of degree 3 and a polynomial p.

We find that the Hermite and the Argyris elements approach their theoretically optimal convergence rate, with an error mildly higher than the Lagrange solution of respectively same degree. It seems like the higher error in interpolation carries over to the boundary value problem, as expected by the definition of the discretization error given by Céa's lemma, even though the difference between the Lagrange element of degree 5 and the Argyris element is smaller than the difference in interpolation.

For an example that involves the boundary condition conforming spaces, consider the Poisson equation with Dirichlet boundary conditions. As discussed in Section 3.3, for the Hermite and the Argyris element, they can be enforced in a strong sense, by constructing a discrete space that obeys the boundary conditions, or in a weak sense via Nitsche's method.

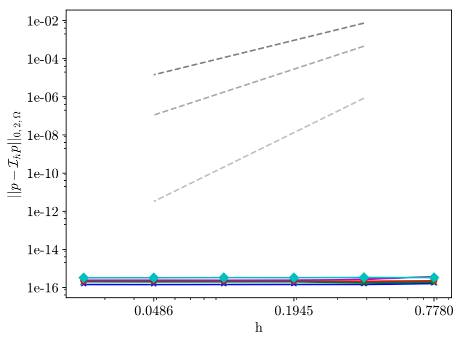
Example 5.2 (Poisson equation with Dirichlet Boundary Conditions). The weak formulation of the Poisson problem with Dirichlet conditions is given by

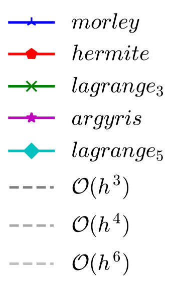
$$\text{Find } u \in \mathrm{H}^1{}_D(\Omega) \text{ such that}$$
$$\int_\Omega \nabla u \nabla v = \int_\Omega fv \quad \forall\, v \in \mathrm{H}^1_0(\Omega), \tag{5.2}$$

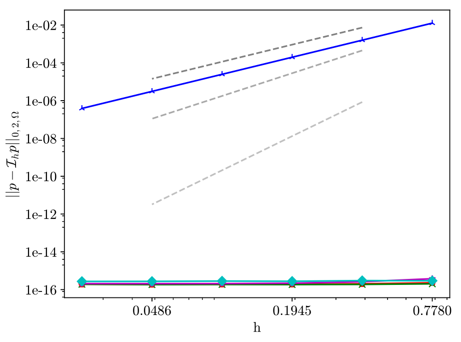
where $\mathrm{H}^1{}_D := \{\, v \in \mathrm{H}^1 : v|_\gamma = u_D\}$. If u solves (5.2), it also solves the Nitsche formulation

$$\text{Find } u \in \mathrm{H}^1(\Omega) \text{ such that}$$
$$\int_\Omega \nabla u \nabla v - \int_\Gamma v \partial_\nu u - \int_\Gamma (u - u_D)\partial_\nu v + \gamma \int_\Gamma (u - u_D)v = \int_\Omega fv \quad \forall\, v \in \mathrm{H}^1(\Omega), \tag{5.3}$$

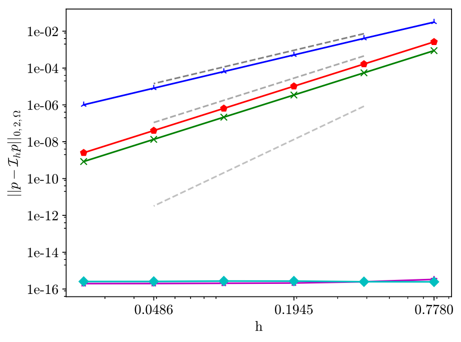
where $\gamma > 0$ is a parameter.
As before, a solution $u(x, y) = x^n + y^n$ is prescribed and the problem to be solved is obtained by

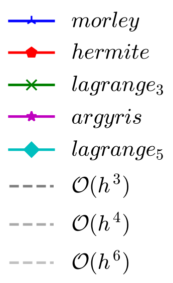


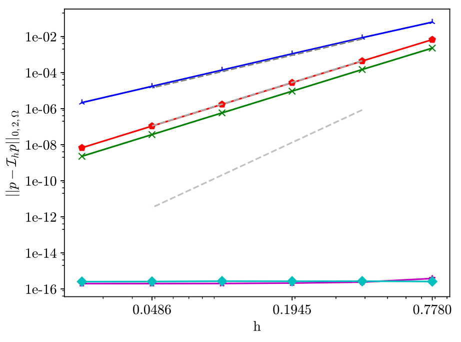




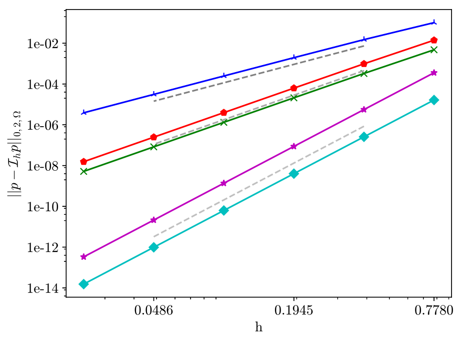
(a) $deg(p) = 2$ (b) $deg(p) = 3$

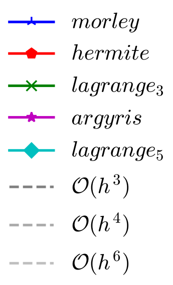


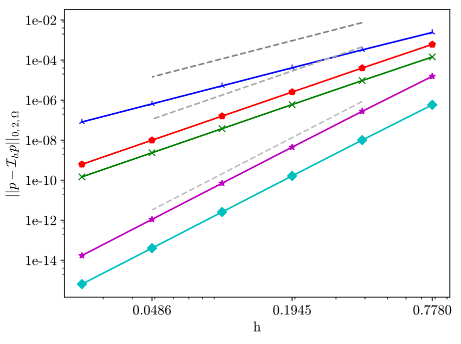




(c) $deg(p) = 4$ (d) $deg(p) = 5$







(e) $deg(p) = 6$ (f) $p(x, y) = \sin(xy)$

Figure 5.3: Interpolation error $\|p - \mathcal{I}_h p\|_{L^2}$ for two-dimensional finite elements. From (a) to (e), p is a polynomial, while for (f) we interpolate a trigonometric function.

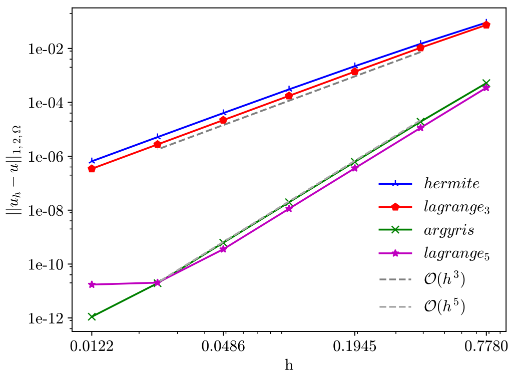
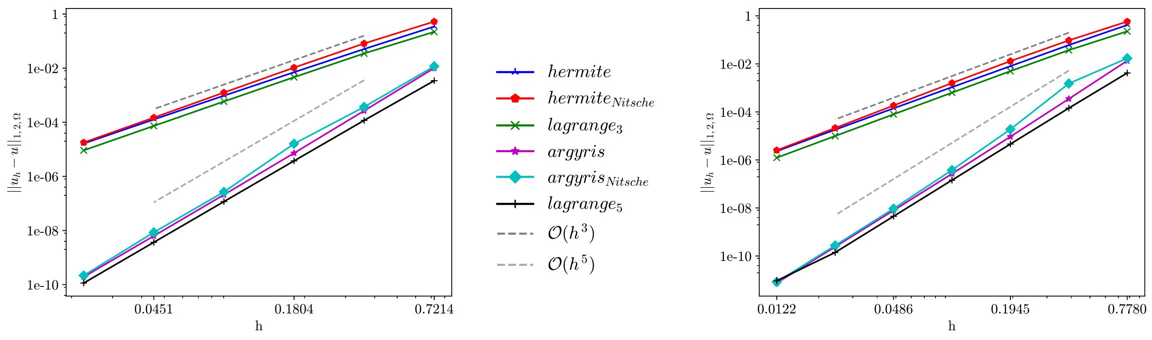


Figure 5.4: H^1-Error for a Reaction-diffusion equation with Neumann boundary conditions



(a) Poisson equation on a square domain.

(b) Poisson equation on the fully perturbed grid. Here the tangential degrees of freedom are used.

Figure 5.5: L^2-Error of two different approaches on Dirichlet boundary conditions. For the Nitsche approach, the parameter η was set to 20.

setting $f(x, y) = -(n-1)n\left(x^{n-2} + y^{n-2}\right)$.

The convergence plot for Example 5.2 solved on the two aforementioned grids is shown in Figure 5.5. Several observations can be made. As a first, this experiment confirms the approach of tangential derivative degrees of freedom, since solving the problem on the fully perturbed grid, where the tangential derivative degrees of freedom are used, does not worsen the convergence qualitatively. Secondly, while the results achieved by Nitsche's method overall show an optimal order of convergence, it seems slightly less stable and in terms of the H^1-error, strongly enforced boundary conditions yield a mildly smaller error. Interestingly, the L^2-error for the Argyris element was slightly smaller for Nitsche's method. However, we omit the plot here. As a third observation, both the Hermite and the Argyris element yield the same order of convergence as the Lagrange elements of respectively same degree, again with a small difference in absolute error in favor of the Lagrange elements (both in terms of the L^2 and H^1 norm).

5.3 Fourth Order Problems

After verifying, that the implemented finite elements can be used for second order boundary value problems, we now turn to fourth order problems, as this is after all the primary use case. Typically, for fourth order problems, one obtains a weak formulation in H^2, and since for $\Omega \subset \mathbb{R}^d, d = 1, 2$, $\mathrm{H}^2(\Omega)$ is compactly embedded in $\mathrm{C}^1(\bar{\Omega})$, one has to use finite elements of class C^1 to construct conforming discrete spaces.

Of the implemented elements, only the one-dimensional Hermite element and the Argyris element for the two-dimensional case are of class C^1 and thus generally suitable for the discretization of fourth order boundary value problems. The Morley element is suitable for a nonconforming discretization for some cases, the example here is the clamped plate problem. However, neither the two nor the three-dimensional cubic Hermite are suitable for higher order problems.

In order to allow a comparison to an alternative approach, we introduce the C^0 *interior penalty approach*. In principle, the idea is to use standard Lagrange finite elements, but to modify the variational problem by including terms that penalize discontinuities of the gradient over the edges of the triangulation. This approach has been analyzed in [Brenner and Sung, 2005] and forms, together with mixed methods, the main alternative to finite elements of class C^1.

5.3.1 C^0 interior Penalty Approach

For completeness, we quote Lemma 5 and its setup from [Brenner and Sung, 2005][1]:

Consider the following variational problem

$$\text{Given } F \in \mathrm{H}^{-1}(\Omega), \text{ find } u \in \mathrm{H}^2_0(\Omega) \text{ such that} \tag{5.4}$$
$$a(u, v) = F(v) \quad \forall\, v \in \mathrm{H}^2_0(\Omega),$$
where
$$a(w, v) = \int_\Omega \left(D^2 w : D^2 v + \beta(x) \nabla w \cdot \nabla v\right) dx \quad \forall w, v \in \mathrm{H}^2(\Omega)$$
$$D^2 w : D^2 v = \sum_{i,j=1}^{2} w_{x_i x_j} v_{x_i x_j}. \tag{5.5}$$

Define the jump of the gradient

$$\left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] := \left.\frac{\partial v_{T_+}}{\partial n_e}\right|_e - \left.\frac{\partial v_{T_-}}{\partial n_e}\right|_e, \tag{5.6}$$

where $e \subset \Omega$ is the shared edge of $T_+ \in \mathcal{T}_h$ and $T_- \in \mathcal{T}_h$ and n_e is the unit normal vector pointing

[1] In fact, Brenner and Sung give a more general version, applicable to polygonal domains. To ease notation, we restrict ourselves to a version only applicable to convex polygonal domains

from T_- to T_+. If $e \subset \Gamma$, define $\left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] := -\frac{\partial v}{\partial n_e}$ with n_e as the unit outward normal vector. Similarly, inside Ω, $\left\{\!\!\left\{\frac{\partial^2 w}{\partial n^2}\right\}\!\!\right\} := \frac{1}{2}\left[\frac{\partial^2 w_{T_+}}{\partial n_e^2} + \frac{\partial^2 w_{T_-}}{\partial n_e^2}\right]$ and for edges on the boundary $\left\{\!\!\left\{\frac{\partial^2 w}{\partial n^2}\right\}\!\!\right\} := \left.\frac{\partial^2 w}{\partial n_e^2}\right|_e$.

Lemma 5.1. *For $F \in \mathrm{H}^1(\Omega)$, the solution u of (5.4) satisfies*

$$\mathcal{A}_h(u, v) = F(v) \quad \forall\, v \in V_h, \tag{5.7}$$

where

$$\mathcal{A}_h(w, v) = a_h(w, v) + b_h(w, v) + \eta c_h(w, v),$$

$$a_h(w, v) = \sum_{T\in\mathcal{T}_h} \int_T \left(D^2 w : D^2 v + \beta(x)\nabla w \cdot \nabla v\right) dx,$$

$$b_h(w, v) = \sum_{e\in\mathcal{E}_h} \int_e \left(\left\{\!\!\left\{\frac{\partial^2 w}{\partial n^2}\right\}\!\!\right\}\left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] + \left\{\!\!\left\{\frac{\partial^2 v}{\partial n^2}\right\}\!\!\right\}\left[\!\left[\frac{\partial w}{\partial n}\right]\!\right]\right) ds,$$

$$c_h(w, v) = \sum_{e\in\mathcal{E}_h} \frac{1}{|e|}\int_e \left[\!\left[\frac{\partial w}{\partial n}\right]\!\right]\left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] ds.$$

Solving (5.4) with a C^0 interior penalty approach then means solving (5.7) using H^1 conforming finite elements. The penalty parameter η has to be chosen, such that the method converges. Its lower bound only depends on the shape regularity of $\mathcal{T}_h$.

These tools at hand, the performance of the implemented C^1 finite elements is now investigated for simple fourth order problems and compared to the penalty approach with Lagrange elements of comparable degree.

5.3.2 Biharmonic Equation

In the strong form, the biharmonic equation is given by

$$\Delta\Delta u = f \quad \text{in } \Omega \tag{5.8}$$

together with suitable boundary conditions.
In the usual manner, multiplication by a testfunction v, integrating over the domain and integration by parts (applied twice) yield the corresponding weak problem

$$\int_\Omega \Delta u \Delta v - \int_\Gamma \Delta u \partial_\nu v + \int_\Gamma v \partial_\nu \Delta u = \int_\Omega f v \quad \forall\, v \in V. \tag{5.9}$$

This problem can be solved for several different types of boundary conditions, two of which are considered here:

1. Simply supported boundary conditions with natural boundary conditions on Δu with bound-

ary data g and h, i.e.,

$$\begin{aligned} u(x) &= g(x), v(x) = 0 & \forall\, x \in \Gamma \,\forall\, v \in V \qquad (5.10)\\ \Delta u(x) &= h(x) & \forall\, x \in \Gamma. \end{aligned}$$

2. Clamped boundary conditions, where both the displacement and rotation at the boundary are fully specified, with boundary data g and h, i.e.,

$$u(x) = g(x), v(x) = 0, \partial_\nu u = h(x), \partial_\nu v = 0 \quad \forall x \in \Gamma \,\forall\, v \in V. \qquad (5.11)$$

Both types of boundary conditions remove the space of linear polynomials from the testspace, which is a necessary condition for the coercivity of the bilinear form.

Example 5.3 (Biharmonic equation with simply supported boundary conditions). For $f(x, y) = 4\pi^4 \sin(\pi x) \sin(\pi y)$,

$$u(x, y) = \sin(\pi x) \sin(\pi y)$$

is the unique solution of Problem 5.9 with inhomogeneous boundary conditions in form of 5.10, where the boundary data is defined by the traces of u.

The convergence plot for Example 5.3 solved on the perturbed grid is depicted in Figure 5.6a.

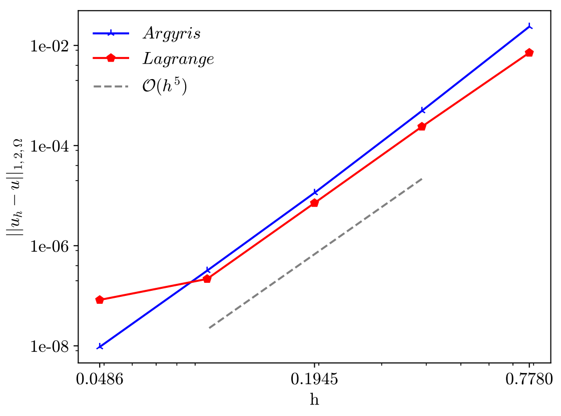
Example 5.4 (Biharmonic equation with clamped boundary conditions). The testproblem with homogeneous clamped boundary conditions in the form of 5.11 and its solution is given by

$$\begin{aligned} \Omega &= (0, 1) \times (0, 1)\\ f(x, y) &= 8\pi^4 \left(1 - 3\sin^2(\pi x) - 3\sin^2(\pi y) + 8\sin^2(\pi x)\sin^2(\pi y)\right)\\ u(x, y) &= \sin^2(\pi x)\sin^2(\pi y). \end{aligned}$$

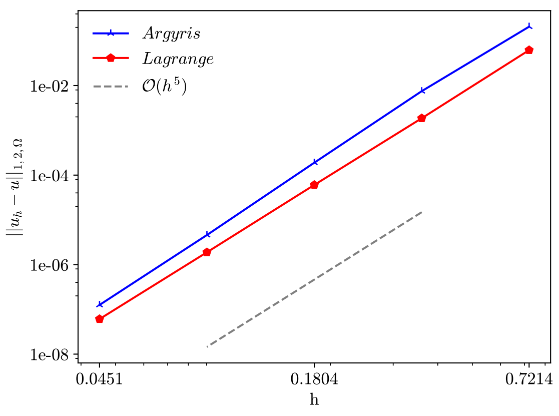
The convergence plot for Example 5.4 solved on the square grid is shown in Figure 5.6b.

For both types of boundary conditions, we note that both the Argyris element and the Lagrange element with penalty approach converge at optimal order, until the result is perturbed by numerical errors. In Figure 5.6a we see that this happens earlier for the Lagrange element than for the Argyris element. This might be due to the high values in the stabilization terms in (5.7). However, it is reasonable to state, that the both methods give qualitatively equal results for these experiments.

As a last example of the biharmonic equation we solve a weak formulation, for which the Morley element provides a suitable nonconforming discretization. To achieve this, we add some terms to (5.9), which sum up to zero when integrated by parts. The modified variational problem



(a) Biharmonic equation with inhomogeneous simply supported boundary conditions.



(b) Biharmonic equation with homogeneous clamped boundary conditions.

Figure 5.6: Convergence behavior for the biharmonic equation with two different boundary conditions.

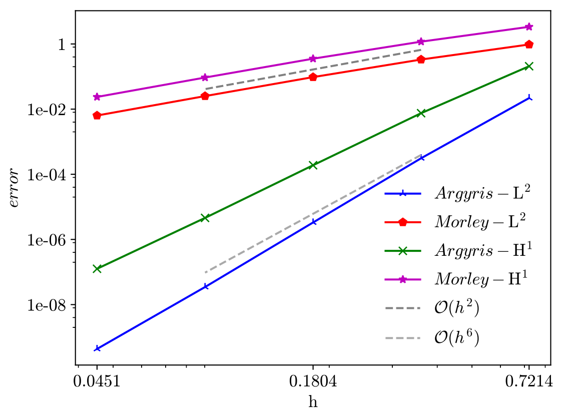
with clamped boundary conditions on Γ reads:

$$\int_\Omega \Delta u \Delta v - (1-\nu) \int_\Omega 2\partial_{xx} u \partial_{yy} v + 2\partial_{yy} u \partial_{xx} v - 4\partial_{xy} u \partial_{xy} v = \int_\Omega fv \quad \forall v \in V.$$

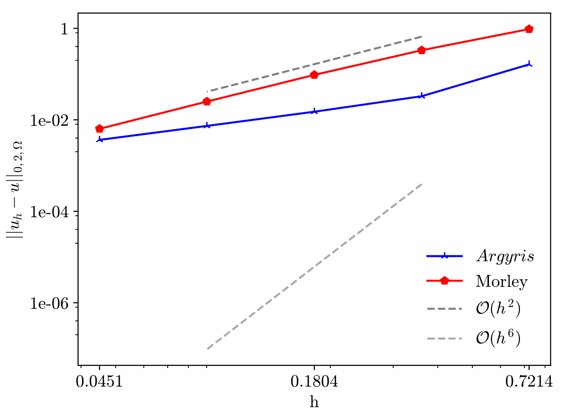
This problem is often called clamped plate problem, as it models the vertical displacement u of a clamped plate, that is, displacement and rotation at the boundary are prescribed, under a force f. Here ν is a physical constant called Poisson's ratio and its physically meaningful values are in $[0, 1/2]$, whereas mathematically we obtain a coercive form for values $0 < \nu < 1$, see [Brenner and Scott, 2002]. We repeat Example 5.4 with the modified form. The convergence plot for the Morley and Argyris element are shown in Figure 5.7a. We see, that L^2 error for the Morley element converges at second order, which is in fact the expected order of convergence ([Kirby, 2018]). However, the H^1 error also converges at second order. To emphasize that for the Argyris element the differentiated treatment of the degrees of freedom at the boundary is necessary, Figure 5.7b shows the lack of convergence if one mistakenly uses all boundary degrees of freedom to incorporate the homogeneous clamped conditions. Note that one could also solve the clamped plate problem with simply supported boundary conditions. However, their natural part would differ from (5.10) by a term induced through the modification.

Runtime comparison

Additional to the convergence behavior of a finite element method, the time spend for computation is also an important factor. In a sequential setup, a finite element method can broadly be split into two parts, firstly the assembling of the linear system and secondly the solving of the same.



(a) Clamped plate problem for $\nu = 0.25$. The Morley element converges at second order for both the L^2 and H^1 error.



(b) Loss of convergence when all boundary degrees of freedom are set to zero.

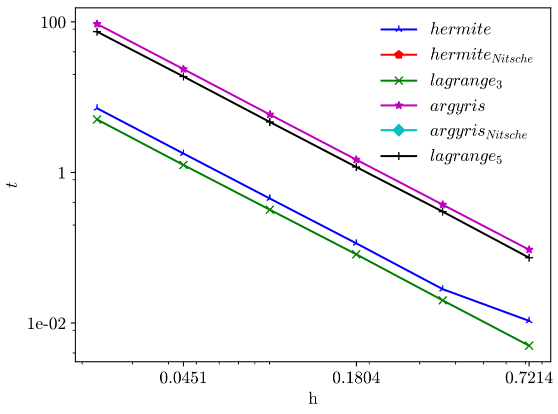
Figure 5.7: Clamped plate problem, solved with Argyris and Morley elements.

5.3.1 Assembling time

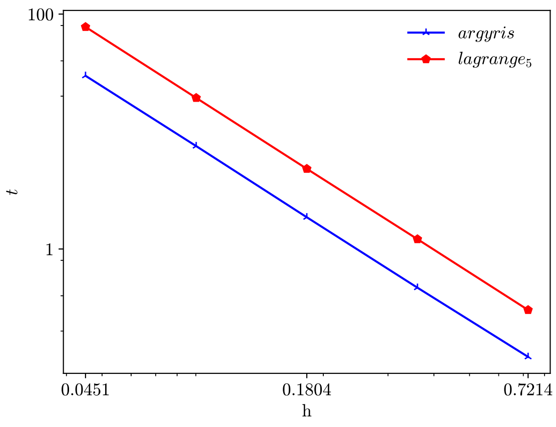
For the comparison of the assembling time for say the Argyris element and the Lagrange element of degree 5 we restrict ourselves to a straight forward implementation of the latter. In practice however, affine equivalence is often exploited further by implementing optimizations like caching of values of basisfunctions or even using a tabulator to precompute all values needed throughout the assembling. Also, we need to distinguish between second and fourth order problems, since for the latter an additional iteration over the edges of the triangulation is involved to assemble the additional values of the C^0 penalty approach. The assembling time for both cases is depicted in Figure 5.8. For second order problems, it shows the expected behavior, namely longer computation time for the linear transformed elements, as the additional transformation simply leads to additional computations. For fourth order problems, we see that the cost of assembling the penalty is higher than the cost of the transformation.

5.3.2 Solving time

A large part of the solving time depends on the size of the assembled matrix, that is, the dimension of the discrete space. Since the Hermite and Argyris element feature multiple degrees of freedom in vertices, for most refinement schemes their global bases should be strictly smaller than a Lagrange basis of corresponding polynomial degree. For the discretizations used throughout this work, this is confirmed in Figure 5.9. While the Morley element has the same amount of basis functions as the Lagrange element of degree 2, the Hermite element has just slightly more than those and stays well below the Lagrange element of degree 3. The Argyris element even has a similar amount of



(a) Time for assembling of a second order problem.



(b) Time for assembling of a fourth order problem.

Figure 5.8: Assembling time for (a) Example 2.19 and (b) Example 5.4. Note that the plot for the fourth order problem contains one refinement step less, but reaches the same order of magnitude.

basis functions as the Lagrange element of degree 3 for finer discretizations and is smaller than the corresponding Lagrange element of degree 5 by about half an order of magnitude.

Note however, that this does not translate completely into runtime improvements, since global basisfunctions located in vertices have a generally larger domain than those located on edges or inside a triangle, which results in a less dense matrix. This is confirmed by Figure 5.10, where this effect is especially visible for the direct solver. When using the iterative solver, the Argyris element seems to be significantly faster than the Lagrange element of degree 5. However, since there are many iterative methods and for this experiment no particular optimization in the choice of solver happened, this result cannot be considered to be generally valid.

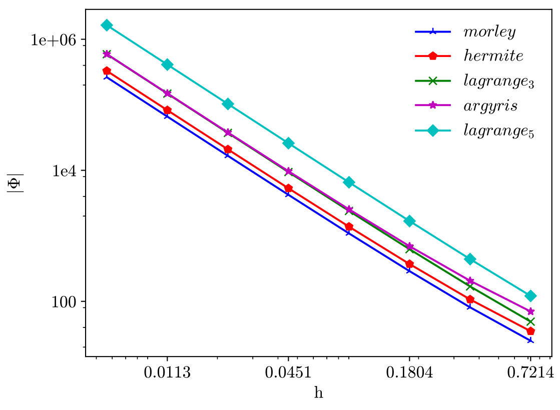


Figure 5.9: Number of global degrees of freedom for different finite element types on a series of red refinements of the grids depicted in Figure 5.1.

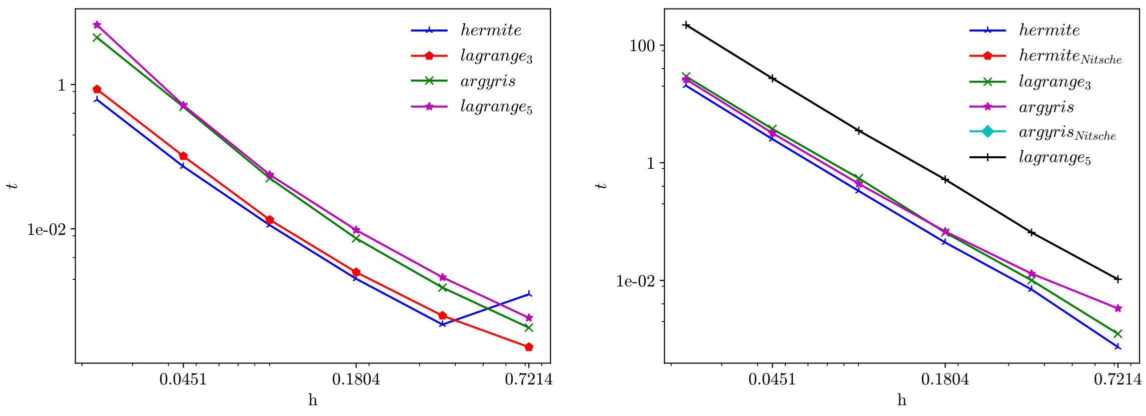


Figure 5.10: Time in seconds for solving the linear equation system resulting from Example 2.19 with a direct solver (left) and an iterative solver (right). The iterative solver is a conjugated gradient descend method with a incomplete LU decomposition as a preconditioner.

6 Föppl-von-Kármán Energy

In this chapter, we discretize the minimization problem for a von-Kármán like energy. In order to minimize this energy, we employ a gradient flow, which leads to two decoupled variational problems at each step, one of them containing a nonlinearity. Throughout this chapter, we follow the discretization strategy and numerical experiments from [Bartels, 2017], who proposed the numerical scheme in combination with Kirchhoff finite elements.

6.1 Problem setting

6.1.1 Variational Problems as Energy Minimization Problems

Many variational problems can be seen, or can be derived, as minimization problems. The quantity to be minimized is called Energy E, is typically given in form of a functional and defined over some suitable functionspace V.
Generalizing the rules from traditional Calculus, the minizer $u := \min_{v \in V} E(v)$ obeys

$$E'(u)[v] = 0 \quad \forall\, v \in V, \tag{6.1}$$

where E' is the Fréchet derivative of E. If E' is linear in both arguments and coercive, (6.1) has the form of a variational problem in the sense of (2.1) and one can readily apply one of the approaches laid out in the previous chapters.
If this is not the case, a common approach is to compute the gradient flow with respect to a scalarproduct $(\cdot,\cdot)_V$ on V:

$$(\partial_t u_t, v)_V = -E'(u_t)[v] \quad \forall\, v \in V. \tag{6.2}$$

Time discretization then leads to a sequence $(u_{t_i})_{i=1,\ldots,n}$ that converges to a local minimum of E, if one exists and the initial solution is sufficiently close.

6.1.2 Von-Kármán Energies

Elasticity is a wide topic with many different models. We will consider an example from the class of plate models, which describe a three-dimensional object, that is much smaller in one dimension

than in the two others, as two-dimensional. They should be contrasted to shell models, which embed the two-dimensional object in the three-dimensional space.
Consider a plate $\omega \subset \mathbb{R}^2$ with thickness $\gamma > 0$. Typically, one is interested in the *deformation*

$$y\colon \Omega_\gamma := \omega \times (-\gamma/2, \gamma/2) \to \mathbb{R}^3$$

that minimizes the elastic energy under a given force f. This energy is assumed to be represented by a stored energy functional W, that, among other physical properties, only depends on the gradients of y. The three-dimensional (hyper) elastic energy can thus be written as

$$E_{3d}(y) = \int_{\Omega_\gamma} W(\nabla y)dx - \int_{\Omega_\gamma} f \cdot y dx.$$

When including some more assumptions, like isotropy of the material, one can obtain more specific energy functionals by rescaling of y and f in terms of γ. For a more detailed justification of the von-Kármán energy (and other energies) we refer to [Ciarlet, 1988, Friesecke et al., 2006].
The energy in plate models generally contains both membrane and bending energies. When considering only one of those, one can eliminate the parameter γ completely. For an example, recall the clamped plate model from Section 5.3, which contains only the bending energy. Von-Kármán theories however, include both types of energy, and thus γ remains as a weight in the sum of two energies. As a consequence, one can observe different phenomena depending on the choice of γ.
Without going further into the details of derivation, we state the energy to be considered:

$$\begin{aligned} E(u,w) = \frac{\gamma^2}{2}\int_\omega \left|D^2 w\right|^2 dx + \frac{1}{2}\int_\omega |\tilde{\varepsilon}(u) + \nabla w \otimes \nabla w|^2 dx \\ - \int_\omega g \cdot u dx - \int_\omega f w dx. \end{aligned} \tag{6.3}$$

Here, $u\colon \omega \to \mathbb{R}^2$ is the in plane *displacement* and $w\colon \omega \to \mathbb{R}$ is the out of plane *deflection*, $D^2 w$ is the Hessian of w, $\tilde{\varepsilon}(u) = \nabla u + \nabla u^T$ twice the symmetric gradient and $a \otimes b = ab^T$ is the dyadic product. The domain of E is a affine space $\mathcal{A} = \mathcal{A}_0 + (u_D, w_D) \subset \mathrm{H}^1(\omega;\mathbb{R}^2) \times \mathrm{H}^2(\omega)$, where $\mathcal{A}_0$ includes the homogeneous boundary conditions, while (u_D, w_D) satisfies the inhomogeneous conditions. As common for plate bending problems, those boundary conditions have to be chosen, such that we have a Korn inequality. In this case, we assume that

$$\|\tilde{u}\|_{\mathrm{H}^1} + \|\tilde{w}\|_{\mathrm{H}^2} \leq C(\|\tilde{\varepsilon}(\tilde{u})\| + \left\|D^2 \tilde{w}\right\|) \quad \forall (\tilde{u}, \tilde{w}) \in \mathcal{A}_0. \tag{6.4}$$

Boundary conditions for which (6.4) holds true are for example clamped conditions on a part of $\partial\omega$ with nonzero surface measure or simply supported boundary condition on the whole boundary, which will be used in the examples in Section 6.2.

6.1.3 Discrete Gradient Flow

The Frechet derivative of E is given by

$$\begin{aligned} E'(u,w)[y,z] =& \gamma^2(D^2w, D^2z) + 2(|\nabla w|^2 \nabla w + \tilde{\varepsilon}(u)\nabla w, \nabla z) - (f,z) \\ &+ (\tilde{\varepsilon}(u), \tilde{\varepsilon}(y)) + (\nabla w \otimes \nabla w, \tilde{\varepsilon}(y)) - (g,y) \end{aligned} \tag{6.5}$$

Given (6.4), it is clear, that $\|(u,w)\|_{\mathcal{A}_0} := \sqrt{\|\tilde{\varepsilon}(u)\|^2 + \|D^2w\|^2}$ is a norm on $\mathcal{A}_0$ and is induced by the scalarproduct $((u,w),(y,z))_{\mathcal{A}_0} := (D^2w, D^2z) + (\tilde{\varepsilon}(u), \tilde{\varepsilon}(y))$. Exploiting the structure of (6.5) one can separate the in plane and out of plane components and yields the gradient flow with respect to $(\cdot,\cdot)_{\mathcal{A}_0}$:

$$\begin{aligned} (D^2\partial_t w, D^2 z) =& -\gamma^2(D^2w, D^2z) - 2(|\nabla w|^2 \nabla w, \nabla z) \\ &- 2(\tilde{\varepsilon}(u)\nabla w, \nabla z) + (f,z), \end{aligned} \tag{6.6}$$

$$(\tilde{\varepsilon}(\partial_t u), \tilde{\varepsilon}(y)) = -(\tilde{\varepsilon}(u), \tilde{\varepsilon}(y)) - (\nabla w \otimes \nabla w, \tilde{\varepsilon}(y)) + (g,y). \tag{6.7}$$

It remains to discretize the evolution. For the scheme proposed in [Bartels, 2017], this is achieved by replacing the time derivatives with difference quotients

$$d_t u^k = \frac{1}{\tau_k}\left(u^k - u^{k-1}\right), \qquad d_t w^k = \frac{1}{\tau_k}\left(w^k - w^{k-1}\right),$$

and by exploitation of some properties of the discrete time derivatives and the structure of (6.6), they yield

$$\begin{aligned} (D^2 d_t w, D^2 z) =& -\gamma^2(D^2 w^k, D^2 z) - 2(|\nabla w^k|^2 \nabla w^k, \nabla z) \\ &- 2(\tilde{\varepsilon}(u^{k-1})\nabla w^{k-1/2}, \nabla z) + (f,z), \end{aligned} \tag{6.8}$$

$$(\tilde{\varepsilon}(d_t u)), \tilde{\varepsilon}(y)) = -(\tilde{\varepsilon}(u^k), \tilde{\varepsilon}(y)) - (\nabla w^k \otimes \nabla w^k, \tilde{\varepsilon}(y)) + (g,y). \tag{6.9}$$

For the details of derivation and concrete properties of the flow we refer to [Bartels, 2017], but note that the flow is unconditionally stable and energy decreasing. The two equations are decoupled in the sense that (6.8) does not involve u^k. Hence, we can compute w^k first and then u^k, which avoids a more complex mixed system.
Lastly it is to be mentioned, that (6.8) contains a nonlinear term and hence a limited number of Newton steps are applied to solve the equation. The Newton iteration causes a mild stability condition on the step sizes τ_k.
Throughout the experiments in the following section, we use the scheme proposed, and deviate from [Bartels, 2017] only in some ways. Firstly, we use the Argyris element presented in the previous

chapters instead of the Kirchhoff element for deflection and a quintic Lagrange element instead of a quadratic Lagrange element for displacement. This removes the need to define a discrete gradient operator, which might lead to additional error sources. Secondly, since the computational effort induced by the high polynomial degree prevents the usage of finer grids, we restrict ourselves with coarser grids. Thirdly, we adapt some parameters, like step size and stopping criterions, which did not allow the computation of relatively exact solutions on coarser grids.

6.2 Numerical Experiments

6.2.1 Experimental Rate of Convergence

First, we want to numerically investigate the performance of the method, in particular in comparison to the implementation with Kirchhoff elements. To do so, we follow example V.2 from [Bartels, 2017], as cited below:

Example 6.1. Let $\gamma = 1, \omega = (0,1) \times (-1/2, 1/2)$ with clamped boundary conditions on $\Gamma_C = \partial\omega$ defined by traces of the functions

$$u(x,y) = \frac{1}{4}\begin{pmatrix} 0 \\ -xy \end{pmatrix}, \quad w(x,y) = \frac{1}{2}x^2 \sin(y).$$

We have

$$g(x,y) = -\frac{1}{2}\left(-1 + 8x\sin^2(y) + 2x^3(\cos^2(y) - \sin^2(y))\right)$$

and

$$\begin{aligned} f(x,y) =& \frac{\gamma^2}{2}\left(-4 + x^2\right)\sin(y) - \frac{1}{4}\left(-4xy\cos(y) + 2(-x + x^3)\sin(y)\right) \\ & - \frac{1}{4}\left(24x^2 - 4x^4\sin^3(y) + (18x^4 - 3x^6)\sin(y)\cos^2(y)\right). \end{aligned}$$

We analyse the errors

$$\delta_u = \|\tilde{\epsilon}(\mathcal{I}_h u - u_h)\|, \quad \delta_w = \left\|D^2(\mathcal{I}_h w - w_h)\right\|$$

for a series of l uniform red refinements of an initial triangulation, which divided ω into two triangles, and for initial solutions given as small perturbation of the exact solutions.

The approximation with Kirchhoff elements showed a linear experimental order of convergence. For the implementation with Argyris elements, the approximation errors are shown in Figure 6.1. We observe an optimal rate of convergence in L^2 norms and δ_w in the first few steps, after which it

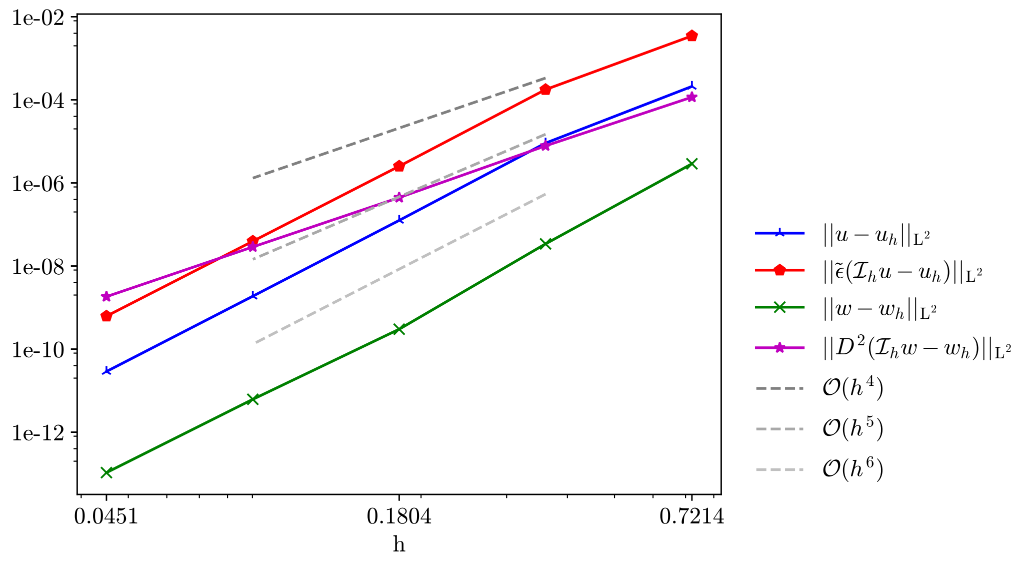


Figure 6.1: Convergence plots for Example 6.1. For the first few triangulations, an almost optimal rate of convergence is observed.

seems like the approximation is perturbed by numerical errors. Particularly interesting is that δ_u also converges at sixth order, which is a priori unexpected. In absolute terms, the errors are much smaller than the ones reported in [Bartels, 2017], which did not drop below 10^{-4} even for finer triangulations. If we consider the approximation with Kirchhoff elements and quadratic Lagrange elements for displacement as an finite element approximation of quadratic degree, then we can still state, that the Argyris element with quintic Lagrange elements yields the same order of convergence relative to the polynomial degree for deflection, and even a better relative convergence behavior for the in plane displacement for this experiment.

6.2.2 Wrinkling Phenomena

As a last application, we try to produce folding and wrinkling patterns by simulating the von-Kármán flow for smaller values of γ. Generally, the bending energy scales with γ^3 while the membrane energy scales with γ^1, see [Friesecke et al., 2006]. For thinner plates and compressive boundary conditions, we expect the emergence of folding and wrinkling phenomena, since membrane energy dominates the bending terms.

We consider the following example, again following [Bartels, 2017]:

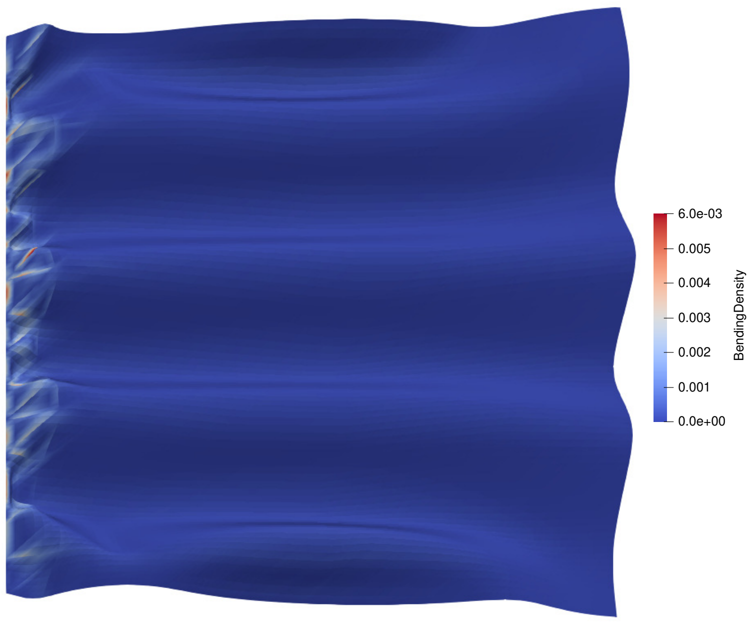
Example 6.2. Let $\omega = (0,1) \times (-1/2, 1/2)$ and $\Gamma_C = \{0\} \times [-1/2, 1/2]$ and set $f = g = 0$ and

$$u_D(x) = \begin{pmatrix} 0 \\ -x_2/10 \end{pmatrix}, \quad w_D(x) = 0 \tag{6.10}$$

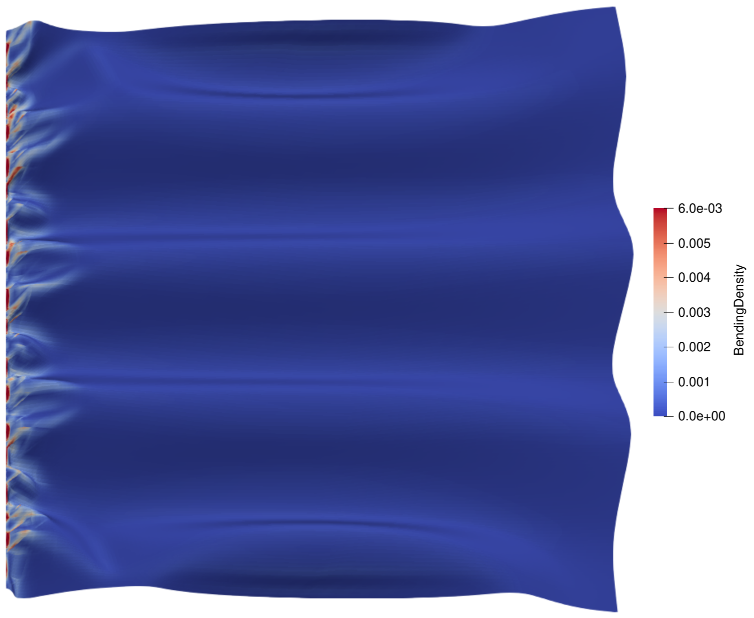
for $x = (x_1, x_2) \in \omega$.
The initial solution is given by the interpolant of

$$\tilde{u}_D(x) = u_D(x), \quad \tilde{w}_d(x) = \frac{1}{2} x_1^2 (1 - x_1^2) \sin(4\pi x_2).$$

The final configuration of the von-Kármán flow defined by Example 6.2 is depicted in Figure 6.2. Similar to the observations in [Bartels, 2017] we observe a dependency on the mesh size. In our case, a mesh size $h_3 \approx 2^{-3}/\sqrt{2}$ seems not sufficiently small to resolve the dynamic induced by the small value of $\gamma = 10^{-3}$. With an additional red refinement, we arrive at a discretization fine enough to produce wrinkling. Compared to the results of the approach based on Kirchhoff elements we note that the computed energy here is one to two orders of magnitude larger than the discretized energy used in [Bartels, 2017]. It is unclear, whether our flow approximates an actually different local minima or this difference is a result of the discretization via Kirchhoff elements and the corresponding discrete gradient. Since the results look qualitatively similar, it is tempting to assume the latter, but a detailed comparison of the computed solutions remains for future work.



(a) For a 3 times refined triangulation the density of the bending energy seems to accumulate around the edges.



(b) On the fourth refinement the edge accumulation is only present at edges very close to the boundary and the expected wrinkles appear close to the compressed boundary before with increasing distance they fade out into a more coarse “folding” pattern.

Figure 6.2: Numerical Deformation for $\gamma = 10^{-3}$ colored by bending energy density in the final configuration of Example 6.2 solved on grids with meshsize (a) $h_3 \approx 2^{-3}/\sqrt{2}$ and (b) $h_4 \approx 2^{-4}/\sqrt{2}$ (bottom).

7 Concluding Remarks

So far, DUNE has lacked some historically famous examples of finite elements of class C^1. We have started to fill this gap by implementing some triangular examples of those, in particular the cubic Hermite element in one dimensions and the quintic Argyris element in two dimensions, which give DUNE users the possibility to solve fourth order problems with conforming finite element methods. Additionally, the Morley triangle can be used for nonconforming methods. The implementation provides special means to incorporate boundary conditions in a strong sense, as well as an adapted global interpolation routine. For both cases, the interfaces of DUNE have been stretched to a certain degree, and it should be noted, that the concrete implementions here might be subject to changes in the future.
The implemented elements have been tested with a number of exemplary problems and the results verified the theoretical convergence behavior throughout. We have also seen, that for fourth order problems in two dimensions the Argyris element provides a qualitatively equal and possible more efficient alternative when compared to the C^0 interior penalty approach utilizing Lagrange finite elements of equal polynomial degree. Of course, this high polynomial degree remains the main disadvantage.
Furthermore, when using the Argyris element to solve an example of high complexity, namely the gradient flow of a von-Kármán energy, we saw that the optimal convergence behavior goes through the two-layered iterative scheme used to simulate the flow and to solve the non-linearity at each step. This underlines the capability of the Argyris element to solve complex problems.

Future Work The module so far contains only the most common examples, namely the quintic Argyris element, the cubic Hermite element, and the quadratic Morley element. All of those can be generalized into families with higher polynomial degrees. In the case of the Morley element there are even several families of Morley-like elements, like the family proposed in [Wang and Xu, 2013], which gives nonconforming finite element spaces for elliptic problems of order $2m$ in $\mathbb{R}^n$ with $n \geq m \geq 1$. A straightforward extension of the work presented here would be to expand the module by implementing at least some members of some of those families. Additionally, so far the Bell element as another famous example of C^1 elements is missing and we have not considered quadrilateral finite elements like the Bogner-Fox-Schmidt element at all.

As a general limitation, the Argyris element does not form hierarchical finite element spaces when constructed on a series of refinements. However, recently a modification of the Argyris element has been proposed in [Carstensen and Hu, 2021], which overcomes this limitation by including an additional basis function at newly created vertices. Implementing this variation would allow using it in multigrid methods, which might help to handle the computational effort resulting from the high polynomial degree.

Finally, there are a number of applications to surface finite element methods to explore. The dune modules `dune-curvedgrid` and `dune-curvedgeometry` form the basis of those methods in the DUNE world, by offering classes representing non-affine mappings for surfaces. Here the surface mapping can be parametrized by finite element functions and one can easily imagine benefits, if such a parametrization is differentiable. Additionally, given such a differentiable parametrization, one could define the finite elements discussed in this work on such a surface. In this case, the major theory of Chapter 3 would go through a generalization of affine equivalence to an equivalence under diffeomorphisms, provided one has a suitable definition for the degrees of freedom, like for example by replacing the normal by conormal derivatives. Technically problematic is the transformation of the Argyris element, as this requires the second derivatives of the mapping, which are not provided by the DUNE interface. Easier to implement is the Morley element, which would give the possibility to implement shell theories. Of course, this would require careful numerical analysis of the properties of such finite elements on surfaces and the convergence of the resulting methods is far from obvious.

Bibliography

[Bartels, 2017] Bartels, S. (2017). Numerical solution of a Föppl–von Kármán model. *SIAM J. Numer. Anal.*, 55(3):1505–1524.

[Bastian et al., 2008a] Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Kornhuber, R., Ohlberger, M., and Sander, O. (2008a). A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part II: Implementation and Tests in DUNE. *Computing*, 82(2–3):121–138.

[Bastian et al., 2008b] Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Ohlberger, M., and Sander, O. (2008b). A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework. *Computing*, 82(2–3):103–119.

[Blatt and Bastian, 2007] Blatt, M. and Bastian, P. (2007). The iterative solver template library. In Kagström, B., Elmroth, E., Dongarra, J., and Wasniewski, J., editors, *Applied Parallel Computing – State of the Art in Scientific Computing*, pages 666–675, Berlin/Heidelberg. Springer.

[Blatt et al., 2016] Blatt, M., Burchardt, A., Dedner, A., Engwer, C., Fahlke, J., Flemisch, B., Gersbacher, C., Gräser, C., Gruber, F., Grüninger, C., Kempf, D., Klöfkorn, R., Malkmus, T., Müthing, S., Nolte, M., Piatkowski, M., and Sander, O. (2016). The Distributed and Unified Numerics Environment, Version 2.4. *Archive of Numerical Software*, 4(100):13–29.

[Brenner and Scott, 2002] Brenner, S. C. and Scott, L. R. (2002). *The mathematical theory of finite element methods*, volume 15 of *Texts in Applied Mathematics*. Springer-Verlag, New York, second edition.

[Brenner and Sung, 2005] Brenner, S. C. and Sung, L.-Y. (2005). C 0 interior penalty methods for fourth order elliptic boundary value problems on polygonal domains. *Journal of Scientific Computing*, 22-23(1-3):83–118.

[Brezzi and Marini, 2013] Brezzi, F. and Marini, L. D. (2013). Virtual element methods for plate bending problems. *Computer Methods in Applied Mechanics and Engineering*, 253:455–462.

[Carstensen and Hu, 2021] Carstensen, C. and Hu, J. (2021). Hierarchical argyris finite element method for adaptive and multigrid algorithms. *Computational Methods in Applied Mathematics*, 21(3):529–556.

[Ciarlet, 1988] Ciarlet, P. (1988). *Mathematical elasticity*. North-Holland Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co, Amsterdam New York New York, N.Y., U.S.A.

[Ciarlet and Raviart, 1972] Ciarlet, P. G. and Raviart, P. A. (1972). General lagrange and hermite interpolation in rn with applications to finite element methods. *Archive for Rational Mechanics and Analysis*, 46:177–199.

[Ciarlet, P G, 1978] Ciarlet, P G (1978). *The finite element method for elliptic problems*. Studies in Mathematics and Its Applications. North-Holland.

[Engwer et al., 2017] Engwer, C., Gräser, C., Müthing, S., and Sander, O. (2017). The interface for functions in the dune-functions module. *Archive of Numerical Software*, Vol 5:No 1 (2017).

[Engwer et al., 2018] Engwer, C., Gräser, C., Müthing, S., and Sander, O. (2018). Function space bases in the dune-functions module.

[Friesecke et al., 2006] Friesecke, G., James, R. D., and Müller, S. (2006). A hierarchy of plate models derived from nonlinear elasticity by gamma-convergence. *Archive for Rational Mechanics and Analysis*, 180(2):183–236.

[Girault and Scott, 2002] Girault, V. and Scott, L. R. (2002). Hermite interpolation of nonsmooth functions preserving boundary conditions. *Math. Comp.*, 71(239):1043–1074.

[Kirby, 2018] Kirby, R. C. (2018). A general approach to transforming finite elements. *The SMAI journal of computational mathematics*, 4:197–224.

[Kirby and Mitchell, 2019] Kirby, R. C. and Mitchell, L. (2019). Code generation for generally mapped finite elements. *ACM Trans. Math. Softw.*, 45(4).

[Nitsche, 1971] Nitsche, J. (1971). 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(1):9–15.

[Sander, 2020] Sander, O. (2020). *DUNE — The Distributed and Unified Numerics Environment*. Springer International Publishing.

[Wang and Xu, 2013] Wang, M. and Xu, J. (2013). Minimal finite element spaces for $2m$-th-order partial differential equations in R^n. *Math. Comp.*, 82(281):25–43.

List of Figures

List of Tables