Multi-mesh reduced-order basis method for finite element analysis

by

ASHKAN AKBARIYEH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2017

To my wife,

Rahil Hosseini,

and my parents,

Daryoush and Soodabeh

ACKNOWLEDGEMENTS

I express my sincere gratitude to my supervising professor Dr. Brian Dennis.

ABSTRACT

Multi-mesh reduced-order basis method for finite element analysis

Ashkan Akbariyeh, Ph.D.

The University of Texas at Arlington, 2017

Supervising Professor: Brian H. Dennis

Reduced order modeling of differential equations parametrized over a parameter space can be used to accelerate optimization and parameter estimation problems. The method of snapshots or reduced order basis is well established among researchers as a tool to build reduced order models of ordinary differential equations. The reduced order basis method has been utilized for numerical solution of parametric PDE problems by researchers in recent years and has many advantages over response surface methods. The application of ROB to finite element analysis has been restricted to using a fixed mesh for snapshots. In this work, a new method is developed for construction of ROB from a set of snapshots defined over various meshes. Consistent inner product is defined for the finite dimensional functional spaces and a new general purpose geometric intersection algorithm is developed to enable the inner product computations for all dimensions. Compatible inner products are used to construct the multi-mesh proper orthogonal decomposition method. The newly developed multi-mesh POD method removes the restriction of fixed mesh from ROB method and it can also be applied outside the context of finite element analysis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Numerical solution of partial differential equations has been a popular research topic over the past decades. The continual interest of scientific community and the ever increasing computational capacities have opened new possibilities in all fields of engineering. Numerical simulations of physical phenomena have become essential to modern engineering. This study focuses on the solution process of partial differential equations using finite element method. The purpose of this work is accelerating the solution process of problems which may require repeated solves. The numerical solution begins with discretization of the differential equation over the domain. Consequently, a system of equations arises. We call the original system of equations the high-dimensional model (HDM). We are after reducing the size of the system of equations to decrease computation time. The main approach in this study is applying reduced-order modeling to Galerkin's finite element. In this method, reduced models are built by projecting the solution space into a lower dimensional space. The new model effectively reduces the number of unknowns. These unknowns are coefficients of the reduced basis and once multiplied by the corresponding base vectors, will construct the solution over the original domain. Construction of such reduced models require considerable computation time where the high-dimensional model is solved multiple times in an offline phase. Once the offline phase is completed, the reduced-order basis method gives us the ability to build an equivalent low-dimensional model for the problem and accelerate solution process.

1.1 Brief survey of state of the art

Reduced-order basis method has been given multiple names by researchers in different fields. The original theory stems from the probability theory and it's application has been expanded to other fields throughout the past decades. The proper orthogonal decomposition (POD) or Karhunen-Loève expansion [1] is a classical tool of probability theory and it states that a random function can be expanded as a series of deterministic functions with random coefficients, so that it is possible to separate the deterministic part from the random one [2]. POD method has been intended to be used as a pattern recognition tool to search for common patterns in experimental data. POD entered mechanical engineering with publications on it's applications in turbulent fluid mechanics. Given an ensemble of patterns, the technique yields an orthogonal basis for the representation of the ensemble [3]. Truncation of the optimal basis provides a way to find optimal lower dimensional approximations of the given data [4]. This optimal quality of POD method has made it a good candidate for data compression and model reduction in various fields of science and engineering. [5] [6] [7] [8] [9] are a few samples from the vast body of research around this subject. In the context of mechanical engineering the POD method has been traditionally used for dynamic systems evolving in time. Combination of POD method with Galerkin's projection is a popular method to build reduced-order models of partial differential equations [10] [11]. Recently the POD method has been applied to build reduced-order models for parametrized optimization purposes [12] [13] [9] [14] [15] [16] [17]. Most of the body of research has been focused on linear differential equations but there are notable publication from multiple groups which have attempted to address the nonlinear case [18] [19] [20] [21] [22] [23]. A note worthy paper by Farhat and Amsallem [24] takes ideas from differential geometry [25] [26] and presents a solid mathematical method for ROM interpolation.

1.2 Research Objectives and Contributions

The focus of this research is reduced-order modeling of finite element problems and extending reduced basis method's application to parametric problems. A parametric model may require different mesh properties for distinct design configurations. The state of the art is that the reduced basis calculated for a specific mesh of the domain can only be used exclusively. A key aspect which I am going to improve is the dependence of the reduced basis vectors on the domain discretization. I focus on providing a method for multi-mesh proper orthogonal decomposition. Reduced-order basis is optimal in the sense that it spans an optimum subspace of certain dimension, given a spanning set of vectors which we call the set of snapshots. In a multi-mesh setting, the snapshot set may be given as combination of vectors defined over a range of different meshes.

The domain of a differential equation can be discretized with arbitrary grids and it desirable to have this freedom. Typically, a reduced-order basis model is created on a fixed grid. At first glance, it may seem that the ROB is applicable only to the same grid but this is not entirely true. It is possible to project each basis vector to the new vector space, i.e. new mesh, however it is no longer guaranteed that the projected basis vectors contain all the information available in their initial form. Having a different mesh other than the one used for creating the ROB, changes the approximation accuracy of the function space. Interpolation methods can be used as an easy way to project vectors between meshes. The more accurate way is to use the inner product operator of the underlying vector space. It has been noted by many other researchers as well that the proper way of projection makes a big difference. I will also provide a substantial mathematical reasoning and formulation to enable

migration of field variables between meshes.

1.3 Background

In this section I provide brief description of my background work in the initial phase of my study that lead to main the idea of this dissertation.

1.3.1 Reduced order modeling in heat transfer

In a collaboration with other researchers, I worked on an inverse parameter estimation case study. The problem involved a moving parametrized heat source acting on a slab of material. The inverse problem was being solved using optimization algorithms and the main goal of this case study was to accelerate the forward solution of the heat transfer problem. The forward solution is required to evaluate the objective function for the optimization algorithm. The objective function was defined to be the norm of difference between the target and current design's temperature distributions. The optimization algorithm would then try to minimize the objective function and hopefully drive it to zero. The problem has been modeled in COMSOL® [27] commercial software for analysis. The model was interfaced with MATLAB® [28] software and the resulting system of equations and load vector was transferred to MATLAB.

My contribution was setting up MATLAB script for parameter estimation and using a reduced order model(ROM) to accelerate the forward problem. I created the ROM using method of snapshots and verified the accuracy of the model. We successfully recovered the parameters characterizing our target temperature distribution using multiple optimization routines provided in MATLAB.

1.3.2 Comparison of reduced order model and response surfaces method

In this work I was building on my previous experience with the reduced-order models. The main goal was to compare ROM against response surface methods. I investigated the cost of building the model as well as its accuracy. I compared ROM with regular multi-variable regression as well as Kriging method. I used DACE [29] toolbox for Kriging response surface method. The ROM was able to achieve much lower error bounds with less model building cost.

CHAPTER 2

REDUCED ORDER MODELING

In scientific computing, we build mathematical model based on phenomenon we want to study and use numerical methods to solve the model. The solution of the model is an approximation to reality. The accuracy of the solution often improves by increasing the level of details in the mathematical model. A mathematical model consists of relationships between mathematical expression. Solving a model involves finding the unknown values for variables reserved in the model.

In the context of finite element analysis, the mathematical model is our chosen finite element formulation for the given problem. We call the unknowns of the finite element model, degrees of freedom (DOF). We control the number of DOFs at discretization step in finite element modeling process. The discretization step is the procedure to convert the geometric representation of the domain to the finite elements which we call their collective a mesh. A more refined mesh results in a more accurate solution. The tendency to create high dimensional models has grown over time as the computer technology progresses.

In the economics of scientific computing, the most important factor is the time required to solve the problem. Increasing DOFs to achieve more reliable results increases the time-cost of the computation. Although new computer technology and better numerical algorithms continuously reduce the time-cost, the appetite for larger models never ends.

In reduced order modeling we attempt to decrease the number of unknowns while maintaining an acceptable level of accuracy. This translates to savings in high resource requirement and long solution time. Once we achieve this goal we call our equivalent problem a reduced-order model (ROM). There are multiple methods that allow us to decrease the degrees of freedom of a problem. Reduced-order models may be developed using different techniques with various accuracy. One might derive a semi analytical solution with fewer unknowns or just simplify governing equation and solve the problem. In this work, I will focus on reduced-order basis method which is based on linear algebra and is suitable for numerical solutions of partial differential equations. The reduced-order basis method digests data generated by high dimensional models to construct it's ROM. The ROM's advantage over a low-dimensional model is its ability to maintain accuracy of the high dimensional analysis while reducing the time-cost significantly.

In reduced-order basis method, we use projector operator which will project our solution space into a lower dimensional subspace. In linear algebra, we can construct projector operators by fixing the basis of both source and target spaces. The challenging step in projection based reduced-order modeling is to find a subspace of the solution space in which we may represent the solution of our problem with the highest accuracy. The most accurate subspace is the whole solution space itself. The reduced-order basis method restricts the solution space hence we can only solve a subset of possible problems. As an example, consider a problem with a particular parametric loading. We have a subset of all possible loads. We form a high dimensional model of the problem and solve for a sampled set of loading cases. After collecting the solution data from the high dimensional model, we apply reduced-order basis method and construct a ROM. The ROM enables us to limit our search for the solution to the

space of admissible functions to the particular parametric loading scenario. Once we build the ROM, it can be used to rapidly solve the problem for new parameter values.

2.1 Preliminary definitions

Let us define an abstract linear differential equation and boundary conditions denoted by $\mathscr{L}$ and $\mathscr{B}$ respectively as

$$\mathscr{L} \in \mathcal{L}(V, W) \quad in\ \Omega$$

$$\mathscr{L}(\boldsymbol{v}) = \boldsymbol{f} \qquad \boldsymbol{v} \in V, \quad \boldsymbol{f} \in W \tag{2.1}$$

$$\mathscr{B}(\boldsymbol{v}) = \boldsymbol{g} \qquad on\ \Gamma \tag{2.2}$$

with the definitions listed bellow

Ω : geometric domain of the problem

Γ : boundary of Ω

V and W : functional vector space defined over domain Ω

$\mathcal{L}(V, W)$: set of all linear differential operators defined over the vector space V

$\mathscr{L}(\boldsymbol{v})$: differential operator of interest

$\mathscr{B}(\boldsymbol{v})$: boundary condition operator over domain boundary Γ

$\boldsymbol{f}$: source term

$\boldsymbol{g}$: prescribed boundary condition

$\boldsymbol{v}$: solution

Note that Γ can be decomposed into non-overlapping parts Γ_D, Γ_N and Γ_R for Dirichlet, Neumann and Robin boundary conditions respectively and equation 2.2 can be defined as a piecewise operator over domain boundary Γ to incorporate all three types of boundary conditions.

Since linear differential operators are a subset of linear transformations, we have no obstacle considering $\mathscr{L}(\boldsymbol{v})$ as a linear map defined in linear algebra. The vector space V is an infinite dimensional vector space. Functional analysis is the extension of linear algebra into infinite dimensional vector spaces. Functional analysis of Hilbert spaces is a good starting point when dealing with differential equations. Consider a second order linear differential equation. The most general solution space will be the set of all functions defined in Ω which are at least twice continuously differentiable which is the set $\{C^2(\Omega), C^3(\Omega), \ldots\}$. It is important to understand that the minimum differentiability requirement for the solution space also depends on the source term function space W. For example, if $W = C^2(\Omega)$ and the source term $\boldsymbol{f} \in W$ is a twice differentiable function, then the solution space V has to be at least 4 times differentiable.

The differential equations are usually defined over a continuous domain(Ω) and the vector space V is a function space defined over that continuous domain. We are after a function $\boldsymbol{v} \in V$ which is the solution of the differential equation. Based on the differential equation we choose a proper function space with certain properties to guarantee the existence of the solution. All of this preliminary analysis has to be done in the context of infinite dimensional function spaces and functional analysis. In numerical analysis practice we begin with a given infinite dimensional function space and apply some form of discretization which will cast our problem in a finite dimensional space and the problem becomes concrete. Once a surrogate finite dimensional problem is formed, we can apply linear algebra and carry on numerical computations.

Recall that our linear differential equation can be interpreted as a linear mapping of vectors. The range of a linear map is the set of all the vectors of the form

$\mathscr{L}(\boldsymbol{v})$ which $\boldsymbol{v}$ belongs to vector space V. The nullspace of $\mathscr{L}$ is the set of all non-zero vectors which are mapped to zero vector. Range and nullspace definition are the following:

$$range(\mathscr{L}) = \{\mathscr{L}(\boldsymbol{v}) \in W : \boldsymbol{v} \in V\} \tag{2.3}$$

$$null(\mathscr{L}) = \{\boldsymbol{v} \in V : \mathscr{L}(\boldsymbol{v}) = 0 \in W\} \tag{2.4}$$

The existence and uniqueness of the solution of a differential equation is guaranteed for all source vectors $\boldsymbol{f}$, if the linear map $\mathscr{L}(\boldsymbol{v})$ is invertible. Based on a theorem in linear algebra, an invertible linear map defined over a finite dimensional vector space is injective and surjective. Injective means there is a one-to-one mapping between $\boldsymbol{v} \in V$ and $\boldsymbol{f} \in W$ and surjective means that for all vectors f there exists a vector $\boldsymbol{v}$ such that $\mathscr{L}(\boldsymbol{v}) = \boldsymbol{f}$. These properties can be defined in mathematical terms for linear maps $\mathscr{L}$ as:

- Surjective: $\exists \boldsymbol{v} \in V : \mathscr{L}(\boldsymbol{v}) = \boldsymbol{f} \qquad \forall \boldsymbol{f} \in W$
- Injective: $\mathscr{L}(\boldsymbol{u}) = \mathscr{L}(\boldsymbol{v}) \implies \boldsymbol{u} = \boldsymbol{v} \qquad \boldsymbol{u}, \boldsymbol{v} \in V$

Note that the injective property is equivalent to having $null\ \mathscr{L} = 0$. The nullspace of differential operator without boundary conditions does not satisfy this condition and this is why there is no unique solution without defining boundary conditions.

A linear differential equation subjected to proper boundary conditions can be defined as the differential operator restricted to a subspace of the original vector space V in which the boundary conditions are satisfied. This subspace is usually called the space of admissible functions. The admissible functions are those functions which satisfy all the BC's.

2.2 Reduced-order Basis Method

So far we have shown that the dimension of the solution vector space of a differential equation is not restricted by nature. As mentioned earlier, in reduced-order basis method we restrict the solution space to a lower dimensional subspace. This reduction will limit the scope of the problem to a smaller subset and the reduced-order model must be only be used in that subset. There are different methods to build the sought after reduced subspace. Mode truncation based on eigenvalues is a popular method and works well but finding eigenvalues and eigenvectors can be difficult sometimes. The method of snapshots is another popular method to build the reduced subspace. In snapshot method, we solve the linear differential equation for multiple loading conditions and accumulate solution vectors in a set called snapshots. Then we find a basis which represents the set of snapshots and use this basis to build a projector operator. In mathematical terms we can describe the procedure as the following:

1. Select desired subspace W_f in the vector space of source terms, $W_f \in W$
2. Find a basis $\Psi = \{\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots\}$ for the subspace W_f such that $span(\Psi) = W_f$
3. Solve $\mathscr{L}(\boldsymbol{v}_i) = \boldsymbol{\psi}_i$ for $\boldsymbol{\psi}_i \in \Psi$ and form snapshots set, $S = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots\}$
4. Find a basis for the subspace which contains S as $\Phi = \{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots\}$, where $S \in span(\Phi)$
5. Form the reduced-order equation by projecting operator $\mathscr{L}$ into Φ

The first step is where we essentially design our reduced-order model. We want to build a model to cover solutions to a range of parameter values or design points. The usual sampling methods like full factorial and Latin hypercube can be used to generate design points. Each design point is a vector of parameter values and we need to solve the HDM. The solutions are expensive and we would like to avoid doing so

many of them. There are more responsible algorithms proposed by other researchers to lower the computational cost. In the above, I assumed the model is being built for a range of different source terms, i.e. loading conditions. In such scenario, each design point corresponds to a source term vector. We can sample the design space with many design points but only calculate the minimum requirement. Step two emphasizes the point that it is only necessary to solve the HDM for a basis of the subspace which contains all the designs. The reason is that in the case of linear differential equations, we have a bijection map on vector spaces. Any subspace of the solution space is mapped to its image in the codomain vector space which is also a subspace. The map is injective and therefore for any source vector described as a linear combination of vectors in codomain, the solution vector is a linear combination of vectors in the domain vector space. All we need to do is to find which vectors in the domain are mapped to the basis of the subspace in the codomain. The fourth step can be done by any method of choice since the basis is arbitrary. It is customary to use proper orthogonal decomposition or singular value decomposition due to their optimal property. We can apply singular value decomposition to the matrix of snapshots as following:

$$S = U\Sigma V^T \tag{2.5}$$

which U is a matrix whose columns form a basis of column space of matrix S and the Σ matrix contains singular values on its diagonal up to the rank of S. The singular value decomposition is intimately connected with eigenvalues and eigenvectors.In fact, one can find the sought after basis using an equivalent formulation by solving an eigenvalue problem. A good understanding of SVD will reveal its optimality. The singular values represent the stretching of the subspace spanned by the matrix S in the direction of the basis vectors. The larger singular values correspond to more dominant basis

vectors. Using this interpretation one can truncate the number of basis vectors with no significant loss in accuracy. After choosing $\Phi \subseteq U$ as our reduced-order basis, we restrict the domain vector space of the linear system by means of projection. The resulting system of equations will have the same size as the reduced-order basis set. The truncated SVD is the solution to the best low-rank approximation of an $m \times n$ matrix A in the least-squares sense [30], i.e,

$$\arg\min_{X \in R_p} \|A - X\|_F^2 , \tag{2.6}$$

where R_p denotes the set of all $m \times n$ matrices with rank p and $\|.\|_F^2$ denotes the Frobenius norm. This result is known as the Eckart-Young-Mirsky theorem [30]. X is a matrix with same dimensions as A. We form X by using the first p singular values and truncating the rest. X is the best rank p approximation of A and the basis set of X is the reduced-basis for our modeling purposes.

where R_p denotes the set of all $m \times n$ matrices with rank p and $\|.\|_F^2$ denotes the Frobenius norm. This result is known as the Eckart-Young-Mirsky theorem [30]. X is a matrix with same dimensions as A. We form X by using the first p singular values and truncating the rest. X is the best rank p approximation of A and the basis set of X is the reduced-basis for our modeling purposes.

In the last step, we project the linear operator into the reduced basis to construct the reduced model. Any source term on the right-hand side will be projected into this subspace before solution. After solving for the unknown coefficient, the solution of the problem can be reconstructed in high dimensional space.

CHAPTER 3

MULTI-MESH REDUCED BASIS METHOD

3.1 Introduction

The reduced basis method has been applied to build reduced order models for solving a large system of equations represented by matrices. Solving differential equations involves discretizing the spatial domain using a numerical method of choice and solving an equivalent finite dimensional system of equations. In general, the solution of the differential equation belongs to an infinite dimensional functional vector space. Hilbert spaces are examples of such infinite dimensional function spaces which are suitable for differential equations.

When we discretize a spatial domain we restrict the solution space into a finite dimensional subspace of the original infinite dimensional vector space. This finite dimensional vector space can be defined as the span of specific basis vectors. In the case of finite element analysis, the basis vectors are the collection of all shape functions defined during domain discretization. Note that shape functions are defined locally but the basis vectors of the function space are defined over the entire domain, therefore we expand the domain of each shape function to include the entire domain. After domain discretization, the space of solution is restricted to a subspace and the solution is a linear combination of a basis of this subspace. The solution space admits infinite sets of basis and for convenience, it is desired to use the basis set resulting from discretization. The basis is simply a set of linearly independent vectors. Since we have a basis we can represent a vector using a unique set of coefficients or coordinates. Let

n be the dimension of the solution subspace, we can define a map between solution subspace and the coordinates in Euclidean space $\mathbb{R}^n$:

$$f : V \to \mathbb{R}^n \tag{3.1}$$

which is a bijection between the two vector spaces. The original function vector space V is usually an inner product space. The inner product definition allows us to measure angles between vectors. Since the $\mathbb{R}^n$ is also an inner product space, by applying a theorem in linear algebra we can state that there exists a map between the two vector spaces which preserves the inner product and therefore they are isomorphic. The fact that the two spaces are isomorphic does not mean that any mapping between them is an isomorphism. We can use the following lemma to define such an isomorphism.

Lemma 1. *If V and W are two finite dimensional inner product spaces with dimension n, the mapping f that maps an orthonormal basis of V to an orthonormal basis of W is an isomorphism.*

$$\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \dots \boldsymbol{a}_n\} : \textit{orthonormal basis of } V \tag{3.2}$$

$$\mathcal{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \dots \boldsymbol{b}_n\} : \textit{orthonormal basis of } W \tag{3.3}$$

$$f : V \to W : \boldsymbol{v} \in V \to f(\boldsymbol{v}) \in W : f(\boldsymbol{a}_i) = \boldsymbol{b}_i \tag{3.4}$$

Once we fix a basis set for a vector space, it is convenient to represent vectors by their coordinates in the respected basis. The coordinates are nothing more than the coefficients of each basis. We can define a mapping

$$f : V \to \mathbb{F}^n \tag{3.5}$$

$$\boldsymbol{v} \in V \to f(\boldsymbol{v}) = \{v_1, v_2, \dots, v_n\} \in \mathbb{F}^n \tag{3.6}$$

which maps vectors v from our function vector space V to their coordinates in $\mathbb{F}^n$ with respect to the fixed basis. It is important to emphasize that the mapping f is not necessarily an isomorphism between inner product spaces V and $\mathbb{F}^n$. Although this mapping is an isomorphism for vector spaces, it is not an isomorphism for inner product spaces. It is worth pointing out that in the special case where the fixed basis of V is an orthonormal basis, the mapping f defined above is automatically an isomorphism. If the mapping f is not an isomorphism, the inner product of V is not preserved under the action of f. This means we cannot substitute inner product of V with the standard inner product definition of $\mathbb{F}^n$. This is a profound result which we must pay special attention to.

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_V \neq \langle f(\boldsymbol{u}), f(\boldsymbol{v}) \rangle_{\mathbb{F}^n} \tag{3.7}$$

When we use finite element we essentially define a basis set for a finite subspace of our functional space. It is important to notice that the basis set resulting from finite element discretization is not necessarily an orthogonal set. The orthogonality of the global basis depends on the underlying shape functions of elements. The shape functions are linearly independent but not necessarily orthogonal. It would be convenient to use the inner product $\langle ., . \rangle_{\mathbb{F}^n}$ for the purpose of projection but unfortunately we are not allowed. This is a very important issue. All the algebraic manipulations for solving a linear system of equations are usually done in $\mathbb{R}^n$ and they all take the Euclidean inner product for granted, but none of these calculations are precisely valid with respect to the inner product of the function vector space. For example, if you solve eigenvalue problem and find an orthogonal set of eigenvectors, the eigenvectors are orthogonal in $\mathbb{F}^n$ but they are not really orthogonal in V. Each eigenvector is a function defined over the domain and it has a coordinate in $\mathbb{F}^n$. Since the coordinate is defined for a non-orthogonal basis of shape functions, the inner product is not

preserved. If $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ are two eigenvectors in $\mathbb{F}^n$, we know from equation (3.1) that $\boldsymbol{u}, \boldsymbol{v} \in V$ are the two corresponding functions in space V. If we substitute these vectors in equation (3.7) we observe that:

$$f(\boldsymbol{u}) \perp f(\boldsymbol{v}) \Rightarrow \langle f(\boldsymbol{u}), f(\boldsymbol{v}) \rangle_{\mathbb{F}^n} = 0 \xrightarrow{3.7} \langle \boldsymbol{u}, \boldsymbol{v} \rangle_V \neq 0 \tag{3.8}$$

We have shown that the direct mapping from shape functions to standard basis of $\mathbb{F}^n$ is not an isomorphism. Lemma 1 suggest that we map an orthogonal basis of V to $\mathbb{F}^n$ which is theoretically possible. We can generate an orthogonal basis by applying Gram-Schmidt procedure on the shape functions but this approach is not computationally feasible. The following lemma gives us another approach to enforce isomorphism.

Lemma 2. *If V is a finite dimensional inner product spaces with dimension n and basis $\Psi = \{\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n\}$, then there exists a modified inner product for $\mathbb{F}^n$ such that V is isomorphic to $\mathbb{F}^n$ under the coordinate map of V to $\mathbb{F}^n$.*

$$\begin{aligned} f : & V \to \mathbb{F}^N : \boldsymbol{\psi}_i \to \boldsymbol{e}_i \quad i = 1, \ldots, n \\ & v \to [v]_\Psi \end{aligned} \tag{3.9}$$

$\boldsymbol{e}_i$ is the standard basis of $\mathbb{F}^n$ and $[\boldsymbol{v}]_\Psi$ is the coordinate of v in basis Ψ. The compatible inner product is defined as:

$$\langle \boldsymbol{v}, \boldsymbol{u} \rangle_V \triangleq \langle [\boldsymbol{v}]_\Psi , G_V [\bar{\boldsymbol{u}}]_\Psi \rangle_{\mathbb{F}^n} \tag{3.10}$$

$$(G_V)_{ij} \triangleq \langle \boldsymbol{\psi}_i, \boldsymbol{\psi}_j \rangle_V \in \mathbb{F}. \tag{3.11}$$

It is worth pointing out that in linear algebra, the matrix G_V defined in Lemma 2 is known as the Gramian of the set of vectors $\{\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n\}$. Lemma 2 describes a

method to define a special inner product for vector space $\mathbb{F}^n$ such that a coordinate mapping between inner product space V and $\mathbb{F}^n$ becomes an isomorphism. We call this special inner product, the consistent inner product. The isomorphism between the inner product spaces V and $\mathbb{F}^n$, allows us to correctly and consistently shift all the computation from V to $\mathbb{F}^n$. The benefit is instead of dealing directly with functions, we deal with numbers.

3.2 Subspace projection

When we discretize a domain, we generate a mesh. Any mesh corresponds to a finite dimensional vector space. Suppose we have multiple meshes of the same domain. Each mesh is a finite dimensional vector space which is itself a subspace of the infinite dimensional function space defined over the domain. The main idea of this dissertation revolves around transferring vectors between two finite dimensional subspaces of a function space. In order to transfer vectors from one subspace to another, we need a proper projection transformation. The projection transformation should be consistent with the inner product of the function space.

Theorem 1. *Suppose V is an infinite dimensional function space with inner product $\langle ., . \rangle_V$. If A and B are finite dimensional subspaces of V with corresponding basis sets $\mathcal{A}$ and $\mathcal{B}$, there exist a projection transformation $P_{\mathcal{BA}}$, which projects vectors from A to B with respect to basis sets $\mathcal{A}$ and $\mathcal{B}$.*

$$A, B \subset V, \qquad dim(A) = m, \qquad dim(B) = n$$

$$\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m\} : \textit{a basis of } A \tag{3.12}$$

$$\mathcal{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\} : \textit{a basis of } B \tag{3.13}$$

$$P_{\mathcal{BA}} : A \to B : \boldsymbol{v}_A \in A \to P_{\mathcal{BA}}[\boldsymbol{v}_A]_{\mathcal{A}} \in B \tag{3.14}$$

$$P_{\mathcal{BA}} = G_{\mathcal{BB}}^{-1} G_{\mathcal{BA}} \tag{3.15}$$

$$(G_{\mathcal{BB}})_{ij} \triangleq \overline{\langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle}_V = \langle \boldsymbol{b}_j, \boldsymbol{b}_i \rangle_V \in \mathbb{F}, \qquad \boldsymbol{b}_i, \boldsymbol{b}_j \in \mathcal{B}, \quad i, j = 1 \ldots n \tag{3.16}$$

$$(G_{\mathcal{BA}})_{ij} \triangleq \overline{\langle \boldsymbol{b}_i, \boldsymbol{a}_j \rangle}_V = \langle \boldsymbol{a}_j, \boldsymbol{b}_i \rangle_V \in \mathbb{F}, \qquad \boldsymbol{b}_i \in \mathcal{B}, \quad i = 1 \ldots n \tag{3.17}$$

$$\boldsymbol{a}_j \in \mathcal{A}, \quad j = 1 \ldots m$$

The matrix $G_{\mathcal{BA}}$ is the most challenging part of the computation in the subspace projection algorithm. Each term of the matrix is an inner product in V which is an infinite dimensional function space. The most common inner product in function spaces is the integration of multiplication of two functions:

$$\langle ., . \rangle_V : V \times V \to \mathbb{F} : \langle \boldsymbol{u}, \boldsymbol{v} \rangle_V = \int \boldsymbol{u} \bar{\boldsymbol{v}} . d\Omega \tag{3.18}$$

$$(G_{\mathcal{BA}})_{ij} = \langle \boldsymbol{a}_j, \boldsymbol{b}_i \rangle_V \tag{3.19}$$

The computation of $G_{\mathcal{BA}}$, demands integrating the multiplication of all combinations of basis functions $\boldsymbol{b}_i \in \mathcal{B}$ and $\boldsymbol{a}_j \in \mathcal{A}$. In the general case the $G_{\mathcal{BA}}$ matrix will be dense and it's computation is not tractable on computers beyond a certain problem size. However, in the context of finite element analysis the basis functions are defined locally for each element and are 0 throughout the rest of domain Ω. The matrix $G_{\mathcal{BA}}$ becomes sparse with limited bandwidth and therefore the computation will be scalable. The first difficulty is finding the sparsity pattern of $G_{\mathcal{BA}}$.

$$S = \{(i, j) \ : \ \langle \boldsymbol{a}_j, \boldsymbol{b}_i \rangle_V \neq 0\} \tag{3.20}$$

The second difficulty is computing the integrals which need to be done on the intersection of overlapping mesh elements, containing basis functions $\boldsymbol{b}_j$ and $\boldsymbol{a}_i$ for $(i, j) \in S$. Chapter 4 of this document is dedicated to this task.

The projection transformation defined in theorem 1 is the most accurate image of vector $\boldsymbol{v}_A$ onto subspace B with respect to inner product $\langle ., . \rangle_V$ of vector space V. In other words the norm $\|\boldsymbol{v}_A - \boldsymbol{v}_B\|$ is minimum which is ideal. It is important to mention that the subspace projection transformation is not necessarily invertible. The subspace projection transformation can be considered as a change of basis between two finite dimensional subspace of an infinite dimensional function space.

Proposition 1. *Suppose V is an infinite dimensional function space with inner product $\langle ., . \rangle_V$, and A and B are finite dimensional subspaces of V with corresponding basis sets $\mathcal{A}$ and $\mathcal{B}$. If the vector $\boldsymbol{v}_A \in A$ is the projection of $\boldsymbol{v}$, then the true projection of $\boldsymbol{v}$ onto B is attainable through the subspace projection defined in theorem 1 only when $\boldsymbol{v} - \boldsymbol{v}_A$ has no component on B.*

3.2.1 Illustrating example

In this section, I will present a simple and yet important example to demonstrate the power and validity of the subspace projection method. Consider a simple one dimensional domain Ω and a function $\boldsymbol{v}(x)$ defined as:

$$\Omega = \{x \in \mathbb{R} : 0 \leq x \leq l\} \tag{3.21}$$

$$\boldsymbol{v}(x) = Sin(\frac{\pi x}{l}) + \frac{1}{2} Sin(\frac{5\pi x}{2l}) \qquad \boldsymbol{v}(x) \in V \tag{3.22}$$

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_V : V \times V \to \mathbb{R} : \int \boldsymbol{u}\boldsymbol{v}.\Omega \qquad \|\boldsymbol{v}\| = \sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle_V} \tag{3.23}$$

Following from theorem 1, I have generated two discretization of the domain Ω to represent the two subspaces A and B. Each discretization is a 1D mesh with linear elements. The two sets of basis $\mathcal{A} = \{\boldsymbol{a}_1, \ldots \boldsymbol{a}_6\}$ and $\mathcal{B} = \{\boldsymbol{b}_1, \ldots \boldsymbol{b}_4\}$ corresponding to subspaces A and B are the finite element shape functions defined over Ω. The basis sets are visualized in figure 3.1 and 3.2.

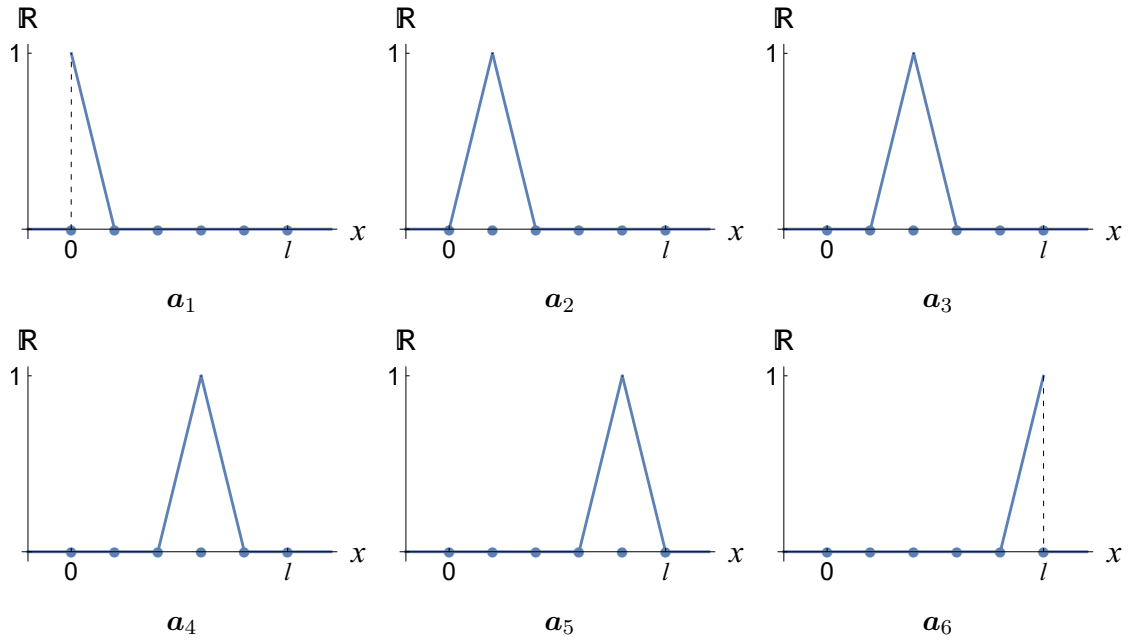


Figure 3.1: Shape functions of mesh A

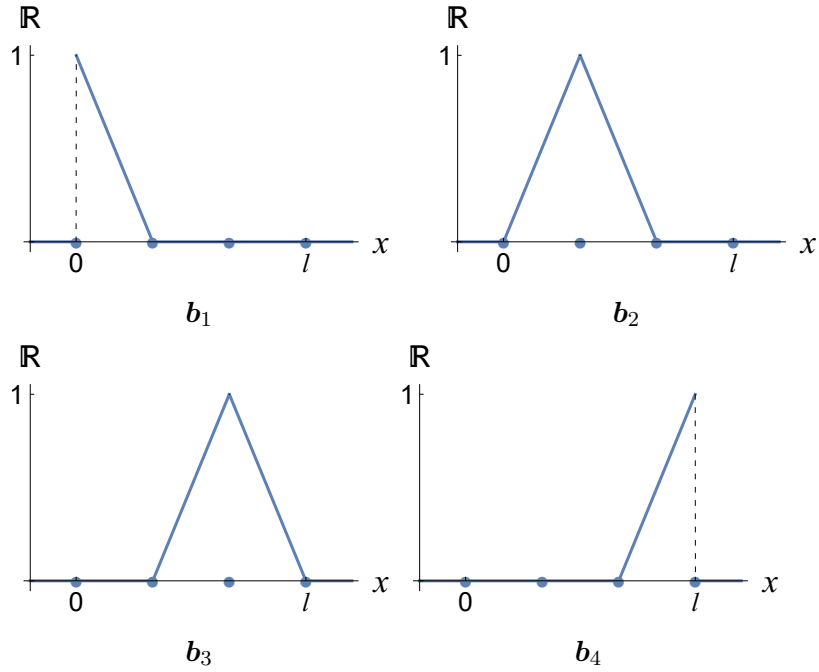


Figure 3.2: Shape functions of mesh B

Proposition 1 states that although the subspace projection method presented here is the optimum projection of $\boldsymbol{v}_A$ onto B, it is not the exact projection of vector $\boldsymbol{v}$ on subspace B. The accuracy of $\boldsymbol{v}_B$ depends on relationship of A and B. The figures 3.3(a) and 3.3(b) show exact projection of $\boldsymbol{v}$ onto the subspaces A and B. The coordinate values of the projections $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ are calculated using the following formulations:

$$\boldsymbol{v} = \boldsymbol{v}_A + \boldsymbol{v}_{A^\perp} = \sum_{j=1}^{6} v_{Aj}\boldsymbol{a}_j + \boldsymbol{v}_{A^\perp} \tag{3.24}$$

$$\langle \boldsymbol{v}, \boldsymbol{a}_i \rangle_V = \langle \sum_{j=1}^{6} v_{Aj}\boldsymbol{a}_j, \boldsymbol{a}_i \rangle_V \qquad i = 1 \ldots 6 \tag{3.25}$$

$$\langle \boldsymbol{v}, \boldsymbol{a}_i \rangle_V = \sum_{j=1}^{6} \langle \boldsymbol{a}_j, \boldsymbol{a}_i \rangle_V v_{Aj} \qquad i = 1 \ldots 6 \tag{3.26}$$

the resulting system of equations is the Galerkin projection of $\boldsymbol{v}$ onto basis $\mathcal{A}$.

$$G_{\mathcal{A}\mathcal{A}}[\boldsymbol{v}_A]_{\mathcal{A}} = [\boldsymbol{f}_v]_{\mathcal{A}} \qquad ([\boldsymbol{f}_v]_{\mathcal{A}})_i \triangleq \langle \boldsymbol{v}, \boldsymbol{a}_i \rangle_V \tag{3.27}$$

similarly for $\mathcal{B}$ we have

$$G_{\mathcal{B}\mathcal{B}}[\boldsymbol{v}_B]_{\mathcal{B}} = [\boldsymbol{f}_v]_{\mathcal{B}} \qquad ([\boldsymbol{f}_v]_{\mathcal{B}})_i \triangleq \langle \boldsymbol{v}, \boldsymbol{b}_i \rangle_V \tag{3.28}$$

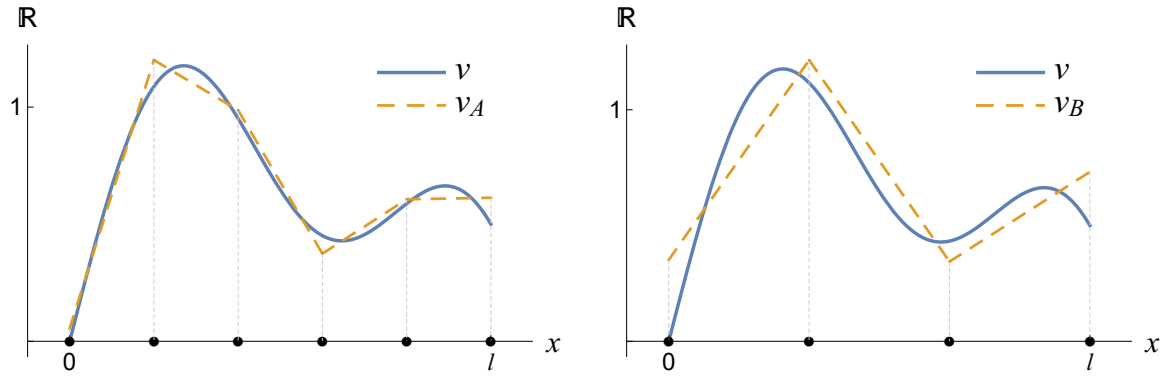
After solving the two systems of equations from above and applying the subspace projection method, the calculated coordinate values will be:

$$[\boldsymbol{v}_A]_{\mathcal{A}} = \{0.0546868, 1.20066, 0.986571, 0.374114, 0.606328, 0.612779\} \tag{3.29}$$

$$[\boldsymbol{v}_B]_{\mathcal{B}} = \{0.351773, 1.21477, 0.344242, 0.731893\} \tag{3.30}$$

$$P_{\mathcal{B}\mathcal{A}}[\boldsymbol{v}_A]_{\mathcal{A}} = \{0.362902, 1.20314, 0.353833, 0.724836\} \tag{3.31}$$

The coordinate values of the projection is close but not equal to the exact projection. This result confirms proposition 1.



(a) (b)

Figure 3.3: Exact projection of function $\boldsymbol{v}$ onto finite dimensional function spaces. (a) Projection onto A. (b) projection onto B

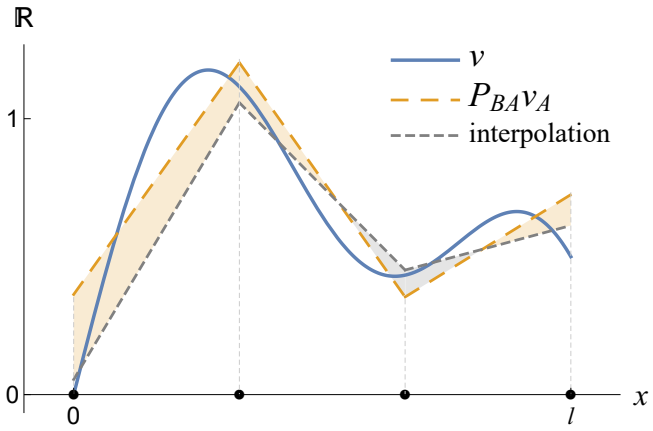


Figure 3.4: Comparison of subspace projection and interpolation. The interpolation is created by sampling $\boldsymbol{v}_A$ on nodal location of B. The shaded area shows the difference between interpolation and projection of $\boldsymbol{v}_A$ onto B.

Figure 3.3 shows two projections of $\boldsymbol{v}$ onto meshes A and B. A naive way to transfer data from mesh A to B is by using interpolation. Figure 3.4 compares interpolation with subspace projection. In this example, the projection of $\boldsymbol{v}_A$ onto B is very close to the exact projection of $\boldsymbol{v}$ onto B. Table 3.1 provides values for norm of various quantities. From the table, we can confirm that the subspace projection is very close to the exact projection since the norm of their differences is very small. Also, we

observe that the subspace projection is far better than the naive interpolation. It is worthy to mention that the subspace projection method presented here can be used in the geometric multigrid method for both prolongation and restriction operations. This example was done using the Wolfram Mathematica ® [31] software.

Table 3.1: Residual norms

$\|\boldsymbol{v} - \boldsymbol{v}_A\|$	0.040303 / l
$\|\boldsymbol{v} - \boldsymbol{v}_B\|$	0.122079 / l
$\|\boldsymbol{v} - P_{BA}\boldsymbol{v}_A\|$	0.122224 / l
$\|\boldsymbol{v} -$ interpolation $\boldsymbol{v}_A\|$	0.190257 / l
$\|\boldsymbol{v}_B - P_{BA}\boldsymbol{v}_A\|$	0.005957 / l

In the following sections of this chapter, I derive multiple methods to compute the proper orthogonal decomposition of a set of vectors defined on multiple meshes. The quality of the methods improve progressively and the final method is the most important contribution. In order to simplify the mathematical notation, without loss of generality I make the assumption that the set S contains n_s snapshots each on a different discretization of the infinite dimensional function space V.

$$V_{h_i} \subset V, \qquad \langle ., . \rangle_{V_{h_i}} : V_{h_i} \times V_{h_i} \to \mathbb{F} \tag{3.32}$$

$$S = \{\boldsymbol{u}_i \mid \boldsymbol{u}_i \in V_{h_i} \ , \ 1 \leq i \leq n_s\} \tag{3.33}$$

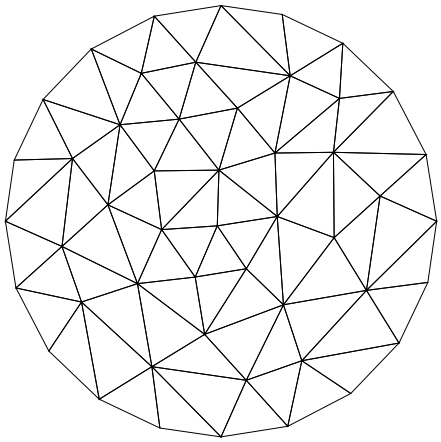
The main goal of the multi-mesh POD methods is to generate reduced basis set for an arbitrary target mesh. I assume we are given a collection of solution snapshots defined on various meshes. The multi-mesh POD methods try to build a reduced basis model using the snapshot set. The problem is non-trivial since the snapshots do not share a common mesh. The snapshots are ultimately functions which are members of function space V. Each snapshot has a different vector representation which is mesh

dependent and it is not possible to simply collect all snapshot vectors in a matrix and apply ordinary POD method to obtain the reduced order basis. The multi-mesh POD methods derivations are based on linear algebra theories and the lemmas and theories proved in this document. The inherent relationship between finite dimensional function spaces V_{h_k} and the encompassing infinite dimensional function space V is exploited to achieve optimal results.

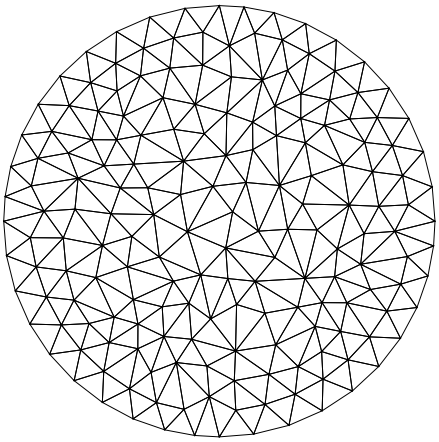
To facilitate the readers understanding, I define a simple problem which is used to make visualizations for the multi-mesh POD methods. Suppose we have four functions defined over unit disk. The functions represent the snapshots in function space V. The snapshots are finite dimensional approximations of the four functions defined on separate meshes. Each mesh, is, in fact, a finite dimensional subspace of V. We desire to obtain reduced basis set defined on an arbitrary target mesh. Figure 3.5 depicts the meshes and table 3.2 provide the mesh properties. Figure 3.6 shows the four snapshot functions defined in functions space V and their finite approximations on each mesh.

	Element type	Order	Nodes	Elements
Mesh 1	Triangle	1	49	76
Mesh 2	Triangle	1	181	320
Mesh 3	Triangle	1	628	1178
Mesh 4	Triangle	1	2515	4876
Target Mesh	Triangle	1	1406	2698

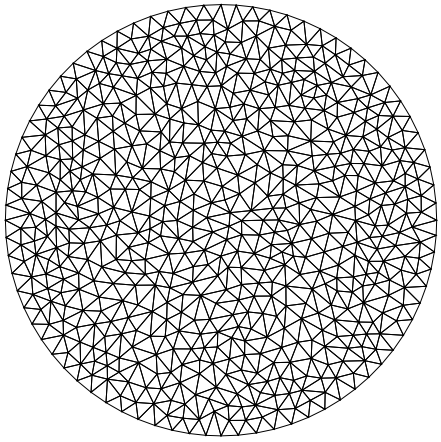
Table 3.2: Mesh properties



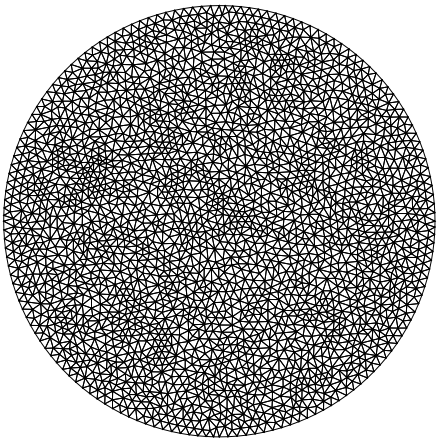
Mesh 1



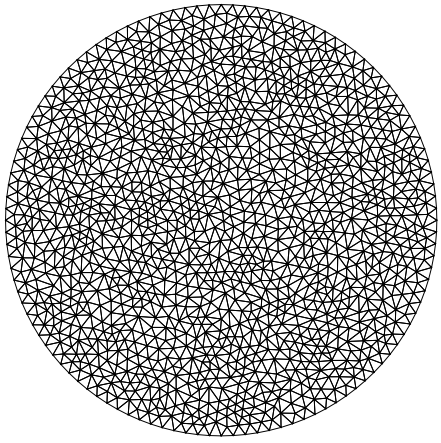
Mesh 2



Mesh 3



Mesh 4



Target Mesh

Figure 3.5: Mesh plots. Meshes 1 to 4 are used for snapshot vectors and the target mesh is reserved for constructing the reduced basis vectors.

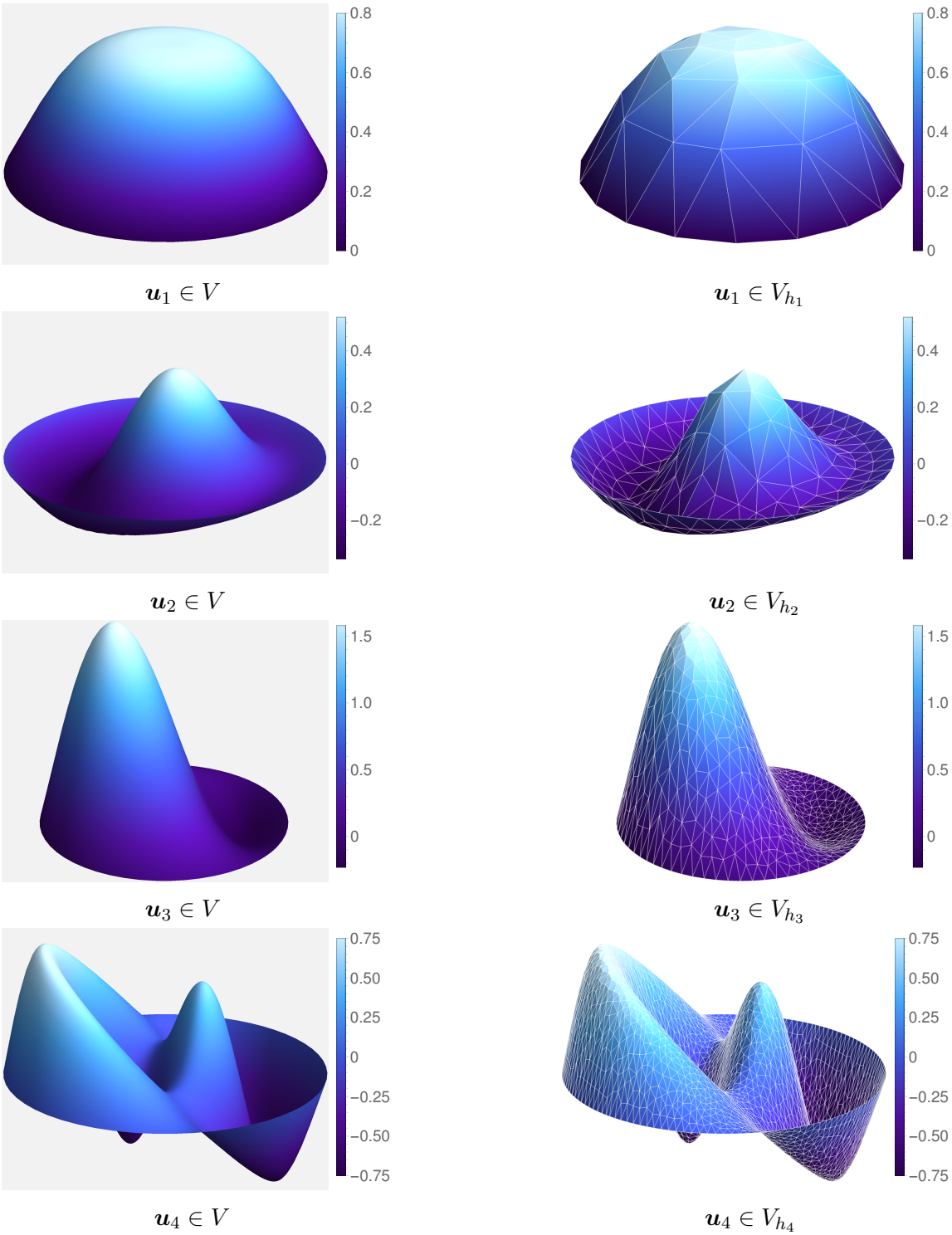


Figure 3.6: Snapshots. On left column we have the true functions belonging to functional space V and on the right column we have their corresponding finite dimensional approximations on snapshots' meshes V_{h_1} to V_{h_4}.

3.3 Method *I*

The main idea of *method I* is to apply proper orthogonal decomposition to separate sets of snapshots independently and combine the basis vectors to form the multi-mesh POD basis on the target mesh.

For each pair of $(\boldsymbol{u}_i, V_{h_j})$, the vector $\boldsymbol{u}_i$ can be uniquely decomposed to components corresponding to projection onto V_{h_j} and $V_{h_j}^{\perp}$. The symbols $\Pi_{V_{h_j}}$ and $\Pi_{V_{h_j}^{\perp}}$ are used to represent these projections in abstract sense before choosing any basis for the function spaces V_{h_i} and V_{h_j}.

$$\boldsymbol{u}_i = \Pi_{V_{h_j}} \boldsymbol{u}_i + \Pi_{V_{h_j}^{\perp}} \boldsymbol{u}_i \qquad 1 \leq i, j \leq n_s \tag{3.34}$$

I define n_s snapshot sets $S(k)$ such that the first set contains the projection of all snapshot vectors $\boldsymbol{u}_i$ onto V_{h_1} and the kth set contains projection of complementary components of $\boldsymbol{u}_i$ from all previous snapshot sets, onto V_{h_k}.

$$S(k) \triangleq \left\{ \Pi_{V_{h_k}} \circ \Pi_{V_{h_{k-1}}^{\perp}} \circ \ldots \circ \Pi_{V_{h_1}^{\perp}} \boldsymbol{u}_j \in V_{h_k} \ : \ j \geq k \right\} \qquad 1 \leq k \leq n_s \tag{3.35}$$

To make it more clear the first two snapshot sets are provided below.

$$S(1) = \boldsymbol{u}_1 \cup \left\{ \Pi_{V_{h_1}} \boldsymbol{u}_i \in V_{h_1} \ : \ i > 1 \right\} \tag{3.36}$$

$$S(2) = \Pi_{V_{h_1}^{\perp}} \boldsymbol{u}_2 \cup \left\{ \Pi_{V_{h_2}} \circ \Pi_{V_{h_1}^{\perp}} \boldsymbol{u}_i \in V_{h_2} \ : \ i > 2 \right\} \tag{3.37}$$

Every $\boldsymbol{u}_i$ vector is sequentially decomposed into multiple components belonging to snapshot sets $S(k)$ with $k \leq i$.

$$\boldsymbol{u}_1 \in S(1) \tag{3.38}$$

$$\boldsymbol{u}_2 = \Pi_{V_{h_1}} \boldsymbol{u}_2 \in S(1) + \Pi_{V_{h_1}^{\perp}} \boldsymbol{u}_2 \in S(2) \tag{3.39}$$

$$\begin{aligned} \boldsymbol{u}_k = \Pi_{V_{h_1}} \boldsymbol{u}_k \in S(1) + \Pi_{V_{h_2}} \circ \Pi_{V_{h_1}^{\perp}} \boldsymbol{u}_k \in S(2) \\ + \ldots + \Pi_{V_{h_{k-1}}^{\perp}} \circ \ldots \circ \Pi_{V_{h_1}^{\perp}} \boldsymbol{u}_k \in S(k) \end{aligned} \tag{3.40}$$

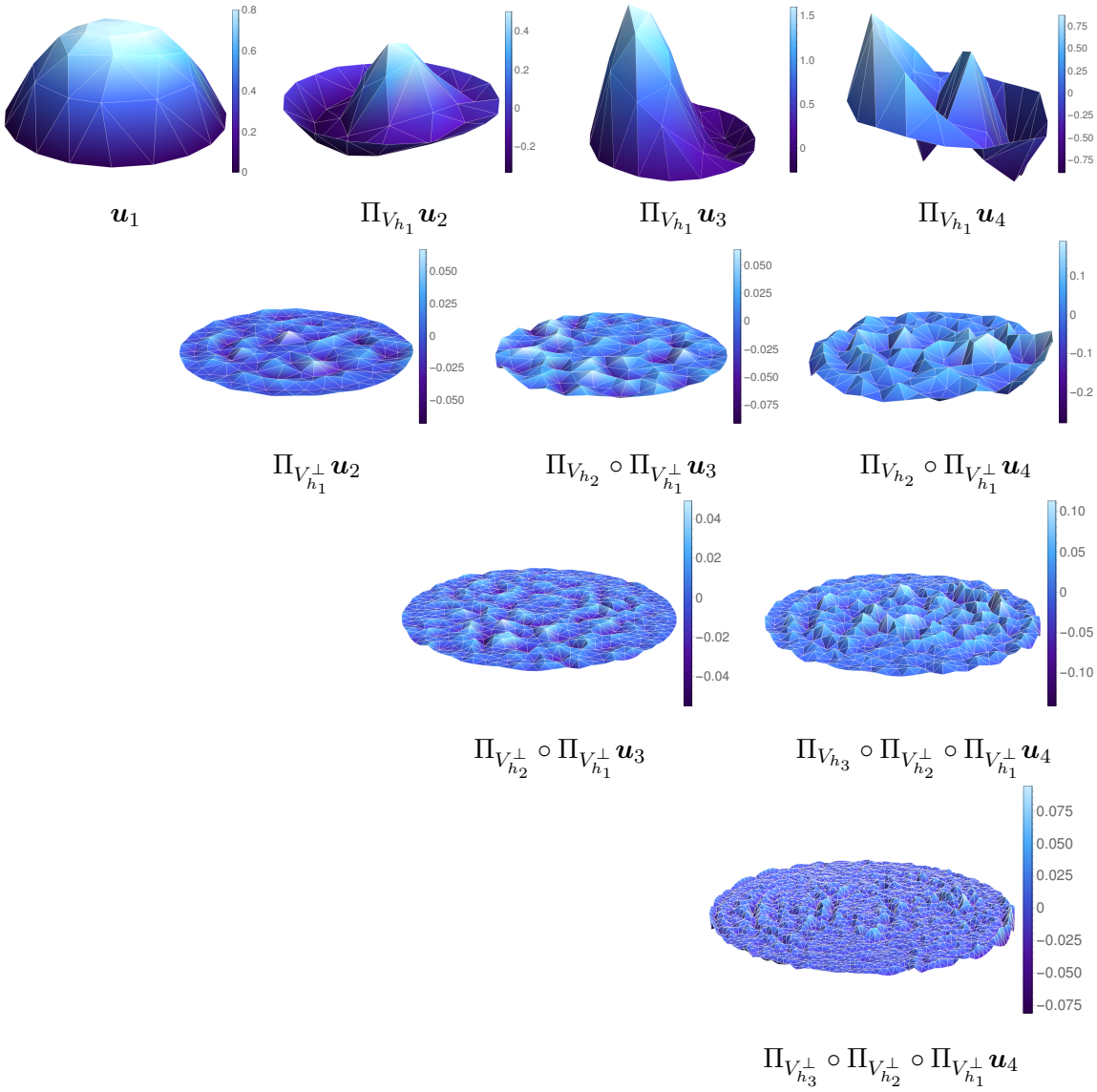


Figure 3.7: Sets of snapshots for method I. From top to bottom, the rows correspond to sets $S(1)$ to $S(4)$. The columns show full decomposition of each snapshot.

By decomposing the snapshot vectors in this manner we avoid using duplicate information. Figures 3.7 shows the snapshots sets $S(k)$ for the example problem. As you can observe, the choice of sequence of processing the meshes will affect the snapshots' decomposition. This is further discussed at the end of this section.

Next we need to apply the POD algorithm with respect to the inner product definition of each function space. Once we choose a basis set such as $\mathcal{H}(k) = \{\boldsymbol{h}_i(k) \ : \ 1 \leq i \leq dim(V_{h_k})\}$ for the function spaces V_{h_k}, the snapshot sets $S(k)$ become matrices. The basis set is nothing but the shape functions of finite element meshes. The consistent inner products can be defined form lemma 2.

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{V_{h_k}} = [\boldsymbol{u}^T]_{\mathcal{H}(k)} G_{V_{h_k}} [\boldsymbol{v}]_{\mathcal{H}(k)}, \qquad (G_{V_{h_k}})_{ij} \triangleq \langle \boldsymbol{h}_j(k), \boldsymbol{h}_i(k) \rangle_V \tag{3.41}$$

We define correlation matrix $C(k)$ and use eigenvalue decomposition to solve for the basis vectors as following:

$$C(k) = S(k)^T G_{V_{h_k}} S(k) \tag{3.42}$$

$$C(k)\boldsymbol{\psi}_i(k) = \sigma_i(k)^2 \boldsymbol{\psi}_i(k) \qquad 1 \leq i \leq r(k) = rank(C(k)) \tag{3.43}$$

$$\boldsymbol{\phi}_i(k) = \frac{1}{\sigma_i(k)} S(k) \boldsymbol{\psi}_i(k) \tag{3.44}$$

$$N(k) = \arg\min_n \frac{\sum_{i=1}^{n} \sigma_i(k)^2}{\sum_{i=1}^{r(k)} \sigma_i(k)^2} \geq 1 - \epsilon_{POD}^2 \tag{3.45}$$

$$\Phi(k) \triangleq \{\boldsymbol{\phi}_i(k) \in V_{h_k} \ : \ 1 \leq i \leq N(k)\} \tag{3.46}$$

The POD basis vectors $\boldsymbol{\phi}_i(k)$ generated by equation 3.46 are plotted in figure 3.8 for the example problem. These vectors can be projected onto any target mesh using the subspace projection method of theorem 1. Take V_h as a target mesh with the basis $\mathcal{H}$. We can compute the POD vectors as following:

$$V_h \in V \qquad \mathcal{H} = \{\boldsymbol{h}_i : 1 \leq i \leq dim(V_h)\} \tag{3.47}$$

$$\tilde{\Phi} \triangleq \left\{ \tilde{\boldsymbol{\phi}}_i = \sum_k \sigma_i(k) P_{\mathcal{H}\mathcal{H}(k)} [\boldsymbol{\phi}_i(k)]_{\mathcal{H}(k)} \ : \ 1 \leq i \leq max(N(k)) \right\} \tag{3.48}$$

$$\Phi \triangleq \{\boldsymbol{\phi}_i : \ 1 \leq i \leq max(N(k))\} = orthonormalize(\tilde{\Phi}) \tag{3.49}$$

where each vector $\tilde{\boldsymbol{\phi}}_i$ is constructed by summation of projection of all $\boldsymbol{\phi}_i(k)$ onto V_h. After recollecting the basis vectors on the target mesh, we orthonormalize the

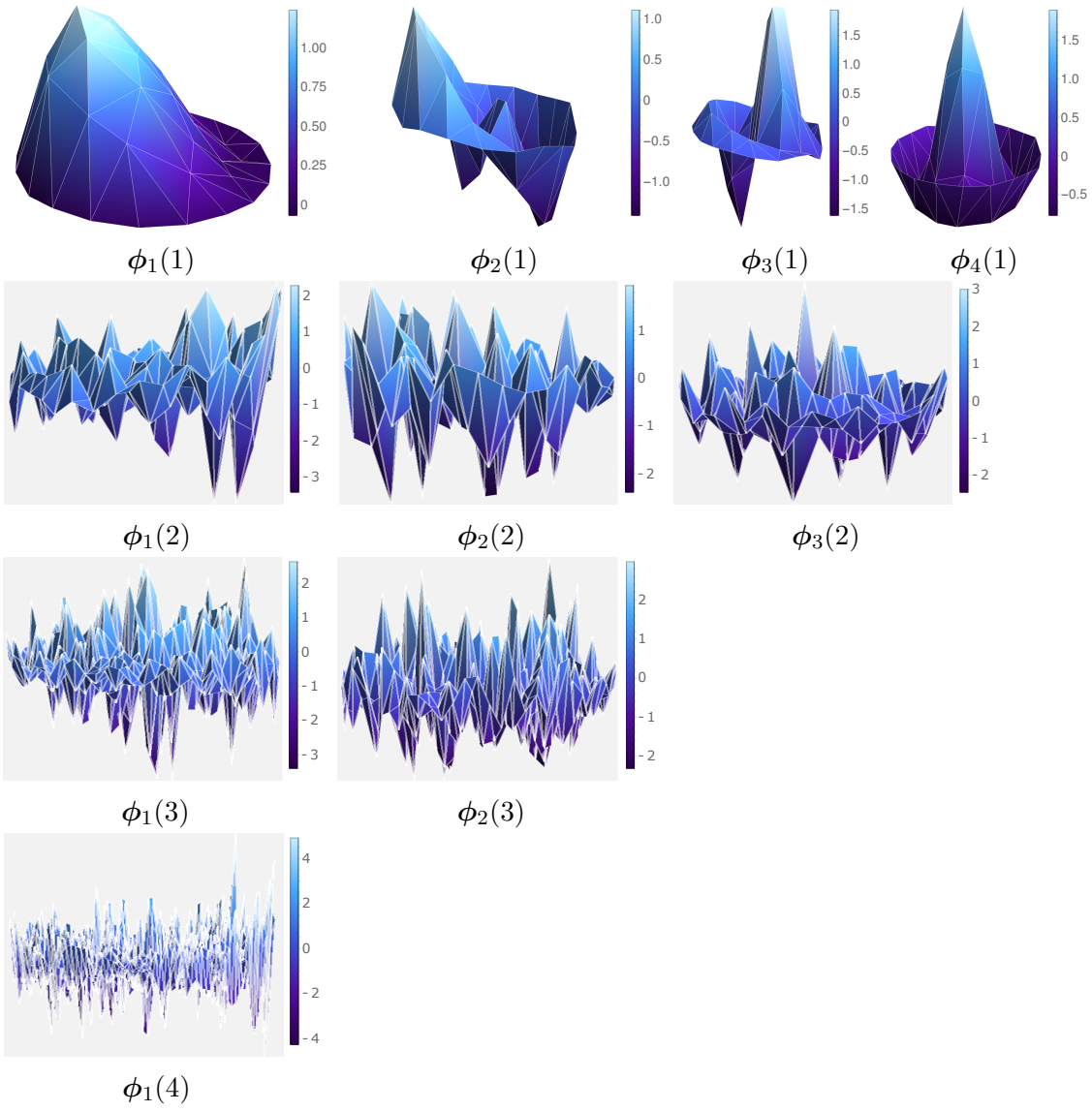
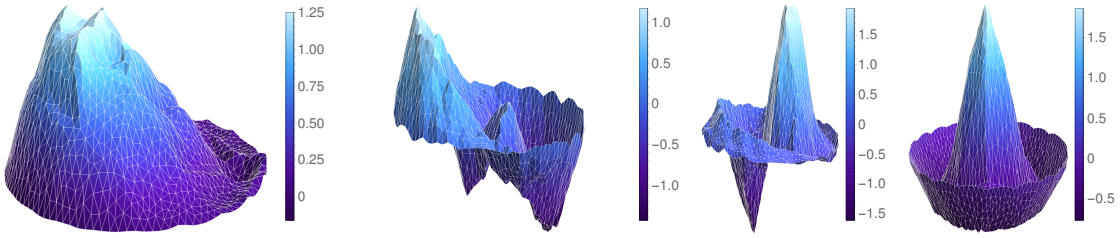


Figure 3.8: POD basis $\Phi(k)$ on snapshots' meshes computed from method I. From top to bottom, the rows correspond to sets $\Phi(1)$ to $\Phi(4)$. The columns show the components of each basis vector on snapshots' meshes.

basis vectors to get the final basis set Φ. During the orthonormalization step, we can take the magnitude of $\tilde{\boldsymbol{\phi}}_i$ vectors as apparent σ_i values. The σ_i values depend on the target mesh since the projection transformations $P_{\mathcal{H}\mathcal{H}(k)}$ may alter the magnitude of $\boldsymbol{\phi}_i$ vectors.



ϕ_1 ϕ_2 ϕ_3 ϕ_4

Figure 3.9: POD basis Φ on target mesh computed from method I

It is worth noting that the basis sets $\Phi(k)$ contain complementary components of global proper orthogonal decomposition of the original snapshot set S. The basis sets are complementary because they represent non redundant components on all function spaces V_{h_k}; However, $span(\bigcup_k \Phi(k)) \neq span(S)$. Figure 3.9 shows projection of ϕ_i vectors onto the target mesh V_h.

For practical computation of the $\boldsymbol{\phi}_i(k)$ basis vectors, we need to provide a concrete procedure of decomposition of snapshot vectors $\boldsymbol{u}_j$. The projection transformation $\Pi_{V_{h_i}}$ can be calculated by the subspace projection method. Recalling the subspace projection formulation, after fixing two basis sets for function spaces $A = V_{h_i}$ and $B = V_{h_j}$, we have:

$$\mathcal{A} : \text{basis set of } V_{h_i} \qquad \mathcal{B} : \text{basis set of } V_{h_j} \tag{3.50}$$

$$[\boldsymbol{u}_j]_{\mathcal{A}} = \Pi_{V_{h_i}} \boldsymbol{u}_j = P_{\mathcal{AB}}[\boldsymbol{u}_j]_{\mathcal{B}} = G^{-1}_{\mathcal{AA}} G_{\mathcal{AB}}[\boldsymbol{u}_j]_{\mathcal{B}} \tag{3.51}$$

From linear algebra we know that any vector can be uniquely decomposed into components belonging to a subspace and it's complement, and also any linear transformation $\mathcal{L}(V, W)$ splits the domain vector space V into $Null(\mathcal{L})$ and quotient space $V/Null(\mathcal{L})$. Taking $\mathcal{L} = P_{\mathcal{AB}}$ we have:

$$V_{h_j} = V_{h_j}/Null(P_{\mathcal{AB}}) \oplus Null(P_{\mathcal{AB}}) \tag{3.52}$$

and we can write $\boldsymbol{u}_j$ as:

$$\boldsymbol{u}_j = \Pi_{V_{h_i}} \boldsymbol{u}_j + \Pi_{V_{h_i}^\perp} \boldsymbol{u}_j = P_{\mathcal{AB}}[\boldsymbol{u}_j]_{\mathcal{B}} + [\tilde{\boldsymbol{u}}_j]_{\mathcal{B}} \tag{3.53}$$

$$\Pi_{V_{h_i}^\perp} \boldsymbol{u}_j = [\tilde{\boldsymbol{u}}_j]_{\mathcal{B}} = \Pi_{Null(P_{\mathcal{AB}})}[\boldsymbol{u}_j]_{\mathcal{B}} \in Null(P_{\mathcal{AB}}) \tag{3.54}$$

For computing the projection onto complement space $\Pi_{V_{h_i}^\perp}$, we need to find the basis for null space of $\Pi_{V_{h_i}}$. LUQ matrix factorization A.2 can be used to obtain a basis for the nullspace of a sparse matrix and it is more efficient than singular value decomposition since it does not generate dense basis vectors; However, the basis vectors are not orthogonal and projection of $\boldsymbol{u}_j$ onto the complement space requires a Galerkin's projection which is equivalent to one linear system solve.

$$Null(P_{\mathcal{AB}}) = Null(G_{\mathcal{AA}}^{-1} G_{\mathcal{AB}}) \tag{3.55}$$

Since $G_{\mathcal{AA}}$ is an invertible matrix we have:

$$dim(Null(G_{\mathcal{AA}}^{-1})) = 0 \implies Null(P_{\mathcal{AB}}) = Null(G_{\mathcal{AB}}) \tag{3.56}$$

which greatly simplifies the task to find the nullspace basis. Suppose that Z is a matrix whose columns form a basis of $Null(P_{\mathcal{AB}})$ computed from the LUQ matrix factorization. The nullspace basis vectors are represented by their coordinates in basis $\mathcal{B}$ of V_{h_j}. For the projection of $\boldsymbol{u}_j$ onto $V_{h_i}^\perp$ using Galerkin's projection we would have:

$$Z = \left\{ [\boldsymbol{z}_i]_{\mathcal{B}} \ : \ [\boldsymbol{z}_i]_{\mathcal{B}} = \sum_{j=1}^{dim(\mathcal{B})} z_{i_j} \boldsymbol{b}_j \ , \ 1 \leq i \leq n_{null} \right\} \tag{3.57}$$

$$[\tilde{\boldsymbol{u}}]_{\mathcal{B}} = Z(Z^T G_{\mathcal{BB}} Z)^{-1} Z^T G_{\mathcal{BB}} [\boldsymbol{u}_j]_{\mathcal{B}} \tag{3.58}$$

As I mentioned earlier, method I for multi-mesh POD decomposes the snapshot vectors and avoids using the duplicate information; However, for arbitrary unstructured meshes, there is no guarantee to use all the information available. This

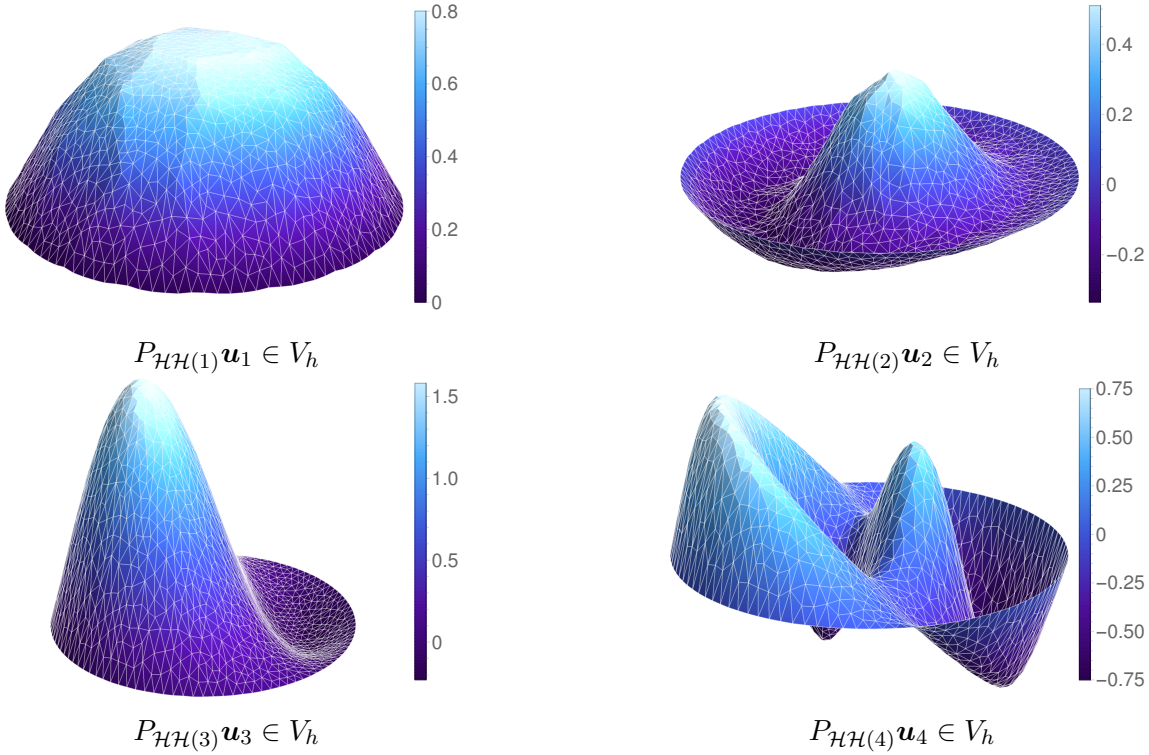
shortcoming stems from the fact that projection transformations between arbitrary non-consistent meshes are not lossless. Every projection between snapshots' meshes may degrade the snapshots. For this reason, the projected POD basis vectors from equation (3.48), lose their orthogonality and we need to apply an orthogonalization procedure such as the Gram-Schmidt. The solution quality of method I also depends on the order we choose to process the snapshots' meshes. Choosing a fine to coarse sequence would yield better results, but this approach does not work all time. For example, we may have multiple fine meshes with different spatial refinements and no matter how we sort the meshes, there is always going to be some information loss.

In method I, I tried to compute POD basis regardless of the target mesh and construct a global reduced order basis by decomposition of snapshots on multiple given meshes. The basis vectors can be computed offline, and for any arbitrary target mesh, we would still need to compute multiple mesh projections. Also, the quality of the result was not perfect due to the abundant use of projection transformations. In the proceeding sections, I provide better solutions to the multi-mesh POD problem.

3.4 Method II

In the second method, we compute the basis vectors after the snapshots are projected onto the target mesh. The main purpose is to avoid losing any data from the snapshots and improve the quality of result compared to method I. Here are the steps required for method II:

1. Compute projection transformation from snapshots' meshes to target mesh.
2. Project all snapshots to the target mesh.
3. Apply proper orthogonal decomposition to the resulting snapshot matrix on the target mesh.



$P_{\mathcal{H}\mathcal{H}(1)}\boldsymbol{u}_1 \in V_h$ $P_{\mathcal{H}\mathcal{H}(2)}\boldsymbol{u}_2 \in V_h$

$P_{\mathcal{H}\mathcal{H}(3)}\boldsymbol{u}_3 \in V_h$ $P_{\mathcal{H}\mathcal{H}(4)}\boldsymbol{u}_4 \in V_h$

Figure 3.10: Snapshots projected onto target mesh

For the first step we use theorem 1 to define the projection transformations. Assuming V_h is the target mesh with basis $\mathcal{H}$ we have:

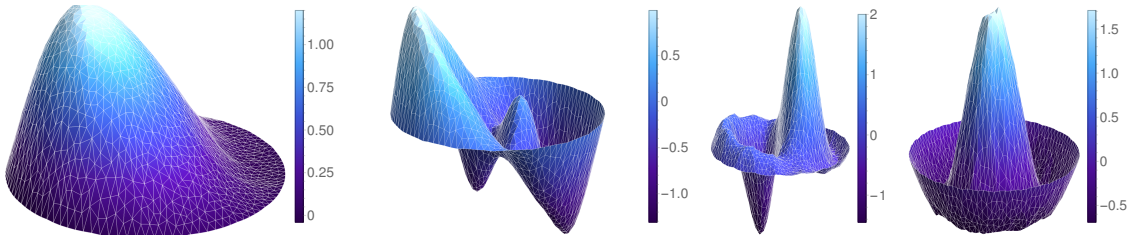
$$P_{\mathcal{H}\mathcal{H}(k)} = {G_{\mathcal{H}\mathcal{H}}}^{-1} G_{\mathcal{H}\mathcal{H}(k)} \qquad 1 \leq k \leq K \tag{3.59}$$

where $\mathcal{H}(k)$ are basis sets of V_{h_k} as before. Next we project all the snapshot vectors $\boldsymbol{u}_k$ onto the target mesh and form new snapshot matrix $[S]_{\mathcal{H}}$. Figure 3.10 shows snapshots projected onto the target mesh V_h.

$$[S]_{\mathcal{H}} = \left\{ P_{\mathcal{H}\mathcal{H}(k)} [\boldsymbol{u}_k]_{\mathcal{H}(k)} : 1 \leq k \leq K \right\} \tag{3.60}$$

In order to compute POD we also need to provide proper inner product definition for V_h using lemma 2.

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{V_h} = \langle [\boldsymbol{u}]_{\mathcal{H}}, G_{\mathcal{H}\mathcal{H}} [\boldsymbol{v}] \rangle_{\mathbb{F}^{dim(V_h)}} \tag{3.61}$$



ϕ_1 ϕ_2 ϕ_3 ϕ_4

Figure 3.11: POD basis Φ on target mesh computed from method II

The POD computation is the following:

$$C = [S^T]_{\mathcal{H}} G_{\mathcal{H}\mathcal{H}} [S]_{\mathcal{H}} \tag{3.62}$$

$$C\boldsymbol{\psi}_i = \sigma_i^2 \boldsymbol{\psi}_i \qquad 1 \leq i \leq r = rank(C) \tag{3.63}$$

$$[\boldsymbol{\phi}_i]_{\mathcal{H}} = \frac{1}{\sigma_i} [S]_{\mathcal{H}} \boldsymbol{\psi}_i \tag{3.64}$$

$$N = \arg\min_n \frac{\sum_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq 1 - \epsilon_{POD}^2 \tag{3.65}$$

$$\Phi \stackrel{\Delta}{=} \{\boldsymbol{\phi}_i \in V_h \ : \ 1 \leq i \leq N\} \tag{3.66}$$

Figure 3.11 shows the final basis vectors computed on the target mesh. Comparing to figure 3.9, the basis vectors are much smoother and the difference is due to superiority of method II over method I in preserving all the information in snapshots.

The advantage of method II over method I is that it is optimal in conserving all available information. The disadvantage is that we need to store all the snapshots. Usually, we have far more snapshots than reduced basis vectors and this will increase computer memory requirement for method II. As far as computation cost, we still need to compute projection transformation between the target mesh and all the snapshot meshes and the POD computation is also differed to the last step which may not be ideal. Although he resulting reduced order model will depend entirely on the target mesh.

3.5 Method III

In the third method, we achieve a truly multi-mesh proper orthogonal decomposition. The basis vectors are computed offline on multiple meshes. We can use the reduced order model as is, or we can choose to project the basis vectors onto an arbitrary target mesh. Here are the steps we take in method III:

1. Define super-mesh as the union of all snapshots meshes.
2. Transfer snapshots to the super-mesh.
3. Compute projection transformations between all snapshot meshes.
4. Form the consistent inner product of super-mesh.
5. Compute POD of snapshots on super-mesh.
6. Compute projection transformation between snapshot meshes and target mesh.
7. Form projection transformation between the super mesh and the target mesh.
8. Project basis vectors to the target mesh.
9. Apply an orthonormalization procedure such as Gram-Schmidt.

Steps 1 to 5 are computed in offline mode. Step 6 which computes the projection transformation between the target mesh and snapshot meshes needs the most computation time in the online step. The immediate advantage over previous methods is that the model reduction is computed offline and we only store the reduced basis vectors versus the entire snapshots. The accuracy of method III is the same as method II for the same target mesh for both methods. However, method III is superior since it works even without a target mesh. There is an opportunity to eliminate the problem of choosing an appropriate target mesh for reduced order models. Even more, interestingly we can devise methods to generate an optimal target mesh for a given problem by mesh fusion.

In the following, I derive the formulation of method III followed by discussing the possibility of solving the reduced order model directly on the super-mesh. Further, I will propose one possible way to fuse the mesh data for an optimal target mesh in the future work section of this document.

Let us begin by introducing the main instrument of method III which is the super-mesh. Intuitively one may want to fuse all the meshes data to form a fully compatible and conforming super-mesh which encompasses all the snapshots' meshes; But this approach requires a very difficult geometric problem to solve and for meshes with mixed order of elements, it is even more difficult to automatically define finite elements with shape functions compatible with various and sometimes customized element definitions coming from multiple meshes. The super-mesh that I define will have many overlapping elements and it can handle mixed and custom finite elements without any complications.

Let the super-mesh $V_{SM} \subset V$ be a finite dimensional function space. Define V_{SM} as the summation of a set of finite dimensional function spaces. Suppose V_{h_k} are snapshots' meshes for $k = 1, \dots K$.

$$V_{SM} = V_{h_1} + V_{h_2} + \ldots + V_{h_K} \tag{3.67}$$

$$N_{sm} = dim(V_{SM}) \leq \sum_{k=1}^{K} dim(V_{h_k}) = N_{tot} \tag{3.68}$$

Similar as before, we take $\mathcal{H}(k)$ to be basis of V_{h_k}. Now we define $\tilde{\mathcal{H}}_{SM}$ to be the spanning set of V_{SM} by taking the union of all $\mathcal{H}(k)$s.

$$\tilde{\mathcal{H}}_{SM} = \left\{ \bigcup_{k=1}^{K} \mathcal{H}(k) \right\} \tag{3.69}$$

From linear algebra, we know that every spanning list in a vector space can be reduced to a basis of the vector space. Suppose that $\mathcal{H}_{SM}$ is $\tilde{\mathcal{H}}_{SM}$ reduced to basis of V_{SM}. The true dimension of V_{SM} is not known before finding $\mathcal{H}_{SM}$; However, we need neither the dimension nor the basis. We can represent any vector in V_{SM} using the spanning set $\tilde{\mathcal{H}}_{SM}$.

$$\forall \boldsymbol{u} \in V_{SM} \qquad \boldsymbol{u} = \sum_{k=1}^{K} \boldsymbol{u}(k) \qquad \boldsymbol{u}(k) \in V_{h_k} \tag{3.70}$$

$$[\boldsymbol{u}]_{\tilde{\mathcal{H}}_{SM}} = [\boldsymbol{u}(1)]_{\mathcal{H}(1)} \frown [\boldsymbol{u}(2)]_{\mathcal{H}(2)} \frown \ldots \frown [\boldsymbol{u}(K)]_{\mathcal{H}(K)} \tag{3.71}$$

Equation (3.71) defines $[\boldsymbol{u}]_{\tilde{\mathcal{H}}_{SM}}$ to be concatenation of of coordinates of $u(k)$s in their respective basis. Although this vector representation is unorthodox since we do not use the basis of V_{SM}, it works in our favor. All of the snapshots are automatically transferred to the super-mesh by zero padding and no information is lost compared to the mesh projection transformations of methods I and II.

$$[S]_{\tilde{\mathcal{H}}_{SM}} = \left\{ [\boldsymbol{u}_k]_{\tilde{\mathcal{H}}_{SM}} : 1 \leq k \leq K \right\} \tag{3.72}$$

After the snapshots matrix $[S]_{\tilde{\mathcal{H}}_{SM}}$ is formed, it only remains to define an equivalent inner product for V_{SM}. The standard inner product of V_{SM} can be defined in $\mathbb{F}^{N_{sm}}$; However since we do not have access to basis of V_{SM}, we define an equivalent inner product in $\mathbb{F}^{N_{tot}}$ using the spanning set $\tilde{\mathcal{H}}_{SM}$:

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{V_{SM}} = \langle [\boldsymbol{u}]_{\mathcal{H}_{SM}}, G_{SM}[\boldsymbol{v}]_{\mathcal{H}_{SM}} \rangle_{\mathbb{F}^{N_{sm}}} = \langle [\boldsymbol{u}]_{\tilde{\mathcal{H}}_{SM}}, \tilde{G}_{SM}[\boldsymbol{v}]_{\tilde{\mathcal{H}}_{SM}} \rangle_{\mathbb{F}^{N_{tot}}} \tag{3.73}$$

$$\tilde{G}_{SM} \triangleq \begin{bmatrix} G_{\mathcal{H}(1)\mathcal{H}(1)} & G_{\mathcal{H}(1)\mathcal{H}(2)} & \ldots & G_{\mathcal{H}(1)\mathcal{H}(K)} \\ G_{\mathcal{H}(2)\mathcal{H}(1)} & G_{\mathcal{H}(2)\mathcal{H}(2)} & \ldots & G_{\mathcal{H}(2)\mathcal{H}(K)} \\ \vdots & \vdots & \ddots & \vdots \\ G_{\mathcal{H}(K)\mathcal{H}(1)} & G_{\mathcal{H}(K)\mathcal{H}(2)} & \ldots & G_{\mathcal{H}(K)\mathcal{H}(K)} \end{bmatrix} \tag{3.74}$$

In equation (3.74), the Gramians $G_{\mathcal{H}(i)\mathcal{H}(j)}$ are defined by equations (3.16) and (3.17) and we have:

$$G_{\mathcal{H}(j)\mathcal{H}(i)} = {G_{\mathcal{H}(i)\mathcal{H}(j)}}^T \tag{3.75}$$

The Gramian computations are the most challenging step of method III since it involves pair-wise mesh intersections. Now that we have the consistent inner product definition, we can apply POD to snapshot matrix $[S]_{\tilde{\mathcal{H}}_{SM}}$ to find the reduced order basis Φ_{SM} over the super-mesh.

$$C = [S^T]_{\mathcal{H}_{SM}} G_{SM} [S]_{\mathcal{H}_{SM}} = [S^T]_{\tilde{\mathcal{H}}_{SM}} \tilde{G}_{SM} [S]_{\tilde{\mathcal{H}}_{SM}} \tag{3.76}$$

$$C\boldsymbol{\psi}_i = \sigma_i^2 \boldsymbol{\psi}_i \qquad 1 \leq i \leq r = rank(C) \tag{3.77}$$

$$[\boldsymbol{\phi}_i]_{\tilde{\mathcal{H}}_{SM}} = \frac{1}{\sigma_i} [S]_{\tilde{\mathcal{H}}_{SM}} \boldsymbol{\psi}_i \tag{3.78}$$

$$N = \arg\min_n \frac{\sum_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq 1 - \epsilon_{POD}^2 \tag{3.79}$$

$$\Phi_{SM} \stackrel{\Delta}{=} \{\boldsymbol{\phi}_i \in V_{SM} \ : \ 1 \leq i \leq N\} \tag{3.80}$$

In equation (3.76), we compute the correlation matrix C using the equivalent inner product definition for vectors described in terms of sapping set $\tilde{\mathcal{H}}_{SM}$ instead of basis set $\mathcal{H}_{SM}$. You can see that we are correctly solving for the POD vectors without sacrificing accuracy. Figure 3.12 visualizes the POD basis vectors as they are defined on the super-mesh. As it can be observed from the figure, each basis vector on the super-mesh may have non-zero components on all snapshot meshes.

After finding the general POD basis on the super-mesh, we need to way to transfer the basis to an arbitrary target mesh. Suppose V_h is a target mesh with basis $\mathcal{H}$. The projection transformation from the super-mesh to the target mesh are computed as a summation of multiple projections from each snapshot's mesh V_{h_k} to

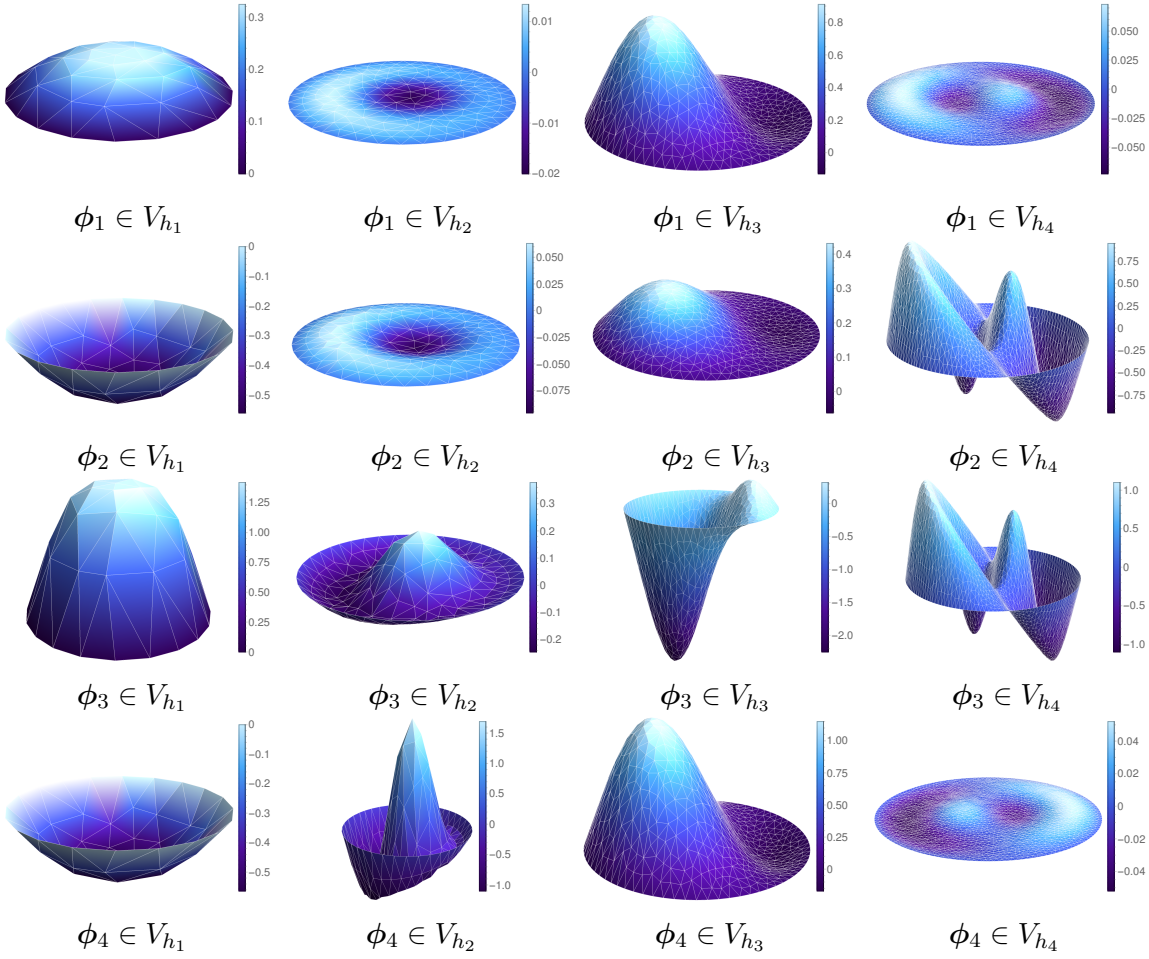
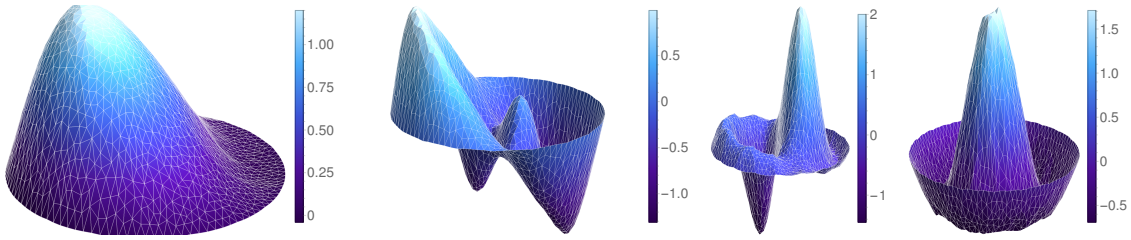


Figure 3.12: POD basis Φ_{SM} on the super-mesh computed from method III. Each row shows components of one basis vector over all snapshot meshes. Each column shows contribution of one mesh to all basis vectors

the target mesh V_h. Figure 3.13 shows the basis vectors on the target mesh after recollection from the super-mesh. The result perfectly matches method II which was ideal.

$$V_h \in V \qquad \mathcal{H} = \{\boldsymbol{h}_i : 1 \leq i \leq dim(V_h)\} \tag{3.81}$$

$$\boldsymbol{u}_{V_h} = P_{\mathcal{H}\tilde{\mathcal{H}}_{SM}}\boldsymbol{u} = G^{-1}_{\mathcal{H}\mathcal{H}} \sum_{k=1}^{K} P_{\mathcal{H}\mathcal{H}(k)}[\boldsymbol{u}(k)]_{\mathcal{H}(k)} \qquad \forall \boldsymbol{u} \in V_{SM} \tag{3.82}$$



ϕ_1 ϕ_2 ϕ_3 ϕ_4

Figure 3.13: POD basis Φ on target mesh computed from method III

Equation (3.83) generates images of basis vectors projected onto the target mesh and similar to method I we need to orthonormalize them. As a reminder, the orthonormalization is required due to the fact that orthogonal projections do not preserve angles between vectors.

$$\tilde{\Phi} \triangleq \left\{ [\tilde{\boldsymbol{\phi}}_i]_{\mathcal{H}} = P_{\mathcal{H}\tilde{\mathcal{H}}_{SM}} [\boldsymbol{\phi}_i]_{\tilde{\mathcal{H}}_{SM}} \ : \ 1 \leq i \leq N \right\} \tag{3.83}$$

$$\Phi \triangleq \{ [\boldsymbol{\phi}_i]_{\mathcal{H}} \in V_h : \ 1 \leq i \leq N \} = orthonormalize(\tilde{\Phi}) \tag{3.84}$$

The super-mesh method has more of potential for application in reduced order modeling which may be explored in future works. As I mentioned earlier, there is an opportunity to eliminate the guess work to determine an appropriate target mesh for a given problem. It might be possible to solve the ROM directly on the super-mesh. To achieve this, one needs to derive formulation equivalent to equations (2.1) and (2.2) for the super-mesh. Being able to directly solve the finite element problem on the super-mesh is highly valuable for multi-mesh ROMs if possible.

Choosing a target mesh for a multi-mesh ROM problem needs some attention as well. In a scenario where the snapshots' meshes have different local refinements, the accuracy of the ROM's solution depends on the target mesh and the error bound might be higher than expected. The number of elements, as well as their density

distribution, affect the solution. To estimate a good target mesh, we can imagine a parametric mesh model which determines the density of ROM's mesh by fusing the original snapshot's meshes. For example, multivariate regression models can be used to correlate parameter values to mesh mixing weights. The mesh weights can then be further processed to generate target mesh density function. The mesh adaptation method pioneered by G. Liao [32] is capable of enforcing mesh density over a previously generated mesh and is a good candidate for the mesh fusion process. If the ROM's solution on the super-mesh is available, we can use the solution decomposition on the snapshots' meshes to determine the mixing weights. The target mesh's density function can be computed by weight normalized linear combination of snapshots' meshes. This method will directly use the solution of the problem whereas the parametric mesh model estimates the density based on the assumption of continuity of the parametric solution manifold and depends on the accuracy of the regression model.

CHAPTER 4

MESH INTERSECTION

Finding the geometric intersection between shapes is a well known mathematical problem. For our purpose, we want to narrow our focus on convex polytopes. Polytopes can be best described as the generalization of polygons from zero dimension to dimensions beyond two. Many variants of shape intersection algorithms exist already but all the methods suffer from some sort of limitation. They are limited either to the spatial dimension or to the types of shapes they can handle. For example, some algorithms are only for finding the intersection between pairs of 2D triangles or pairs of 3D tetrahedrons. Some more general algorithms work for pairs of convex polygons. Although the problem of geometric intersection may not sound mathematically challenging, there is room to improve and generalize existing methods. The limitation of current computational methods to fixed dimension or shape types motivated me to work on a more general algorithm to handle intersection of convex polytopes.

4.1 Embedded convex polytope intersection algorithm

The majority of intersection algorithms work for shapes defined in the same dimension. The embedded convex polytope intersection (ECPI) algorithm is designed to handle any combination of convex polytopes which are embedded in spatial dimension greater or equal to the dimension of the polytopes themselves. The generality of the ECPI algorithm makes it distinguishable and unique. For the purpose of calculating mesh intersections, we need to calculate the intersection between the mesh elements which are polygons in two dimensions or polyhedrons in three dimensions.

Dim	Name	Facet	Ridge	Peak
0	Vertex	Null	–	–
1	Edge	Vertex	Null	–
2	Face	Edge	Vertex	Null
3	Cell	Face	Edge	Vertex
⋮	⋮	⋮	⋮	⋮
n	n-polytope	$(n-1)$-face	$(n-2)$-face	$(n-3)$-face

Table 4.1: Structure of polytopes

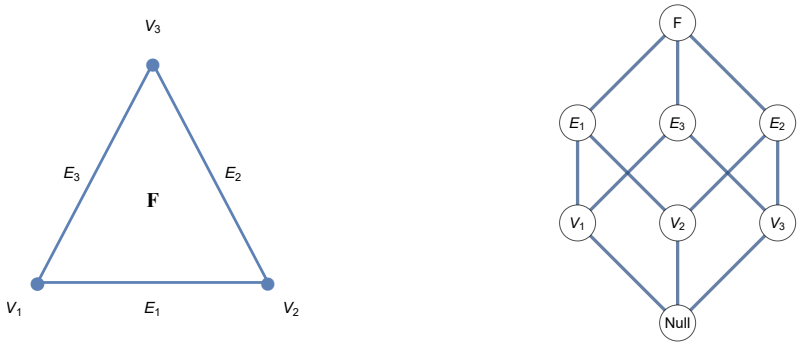
The ECPI algorithm's capability to compute the intersection between polytopes of mismatching dimension is far more than what is required to solve for the intersection between meshes of the same dimension. There are more geometric applications that one can imagine for such a general algorithm. For example, the ECPI algorithm can handle the intersection of an embedded three-dimensional surface and a three-dimensional volume. And the same code can handle intersection of lines and two-dimensional shapes. Examples beyond three dimensions are yet to be tested.

Table 4.1 lists different polytopes and their related facets, ridges and peaks where it applies. The table is provided to establish the terminology used through out this document. Each polytope consists of lower dimensional members. The most immediate sub-geometries are called facets. For example, vertices are facets of edges and edges are facets of faces. From this definition, we can imagine a recursive structure which defines an n-polytope. Each n-polytope contains facets which are $(n-1)$-polytopes and this recursion continues until we reach vertices. An abstract polytope is the pure combinatorial definition which describes the polytope without any geometric representation. In a sense, abstract polytope describes the relationships between various structural elements of a group of polytopes. Each geometric polytope is a realization of its associated abstract polytope in some real N-dimensional space.

The geometric shape of each element type in finite element method can be described by a unique abstract polytope. In fact, the connectivity diagram of an element definition is equivalent to an abstract polytope definition. An abstract polytope can be visualized by Hasse diagrams. Figures 4.1 to 4.5 depict Hasse diagrams for conventional finite elements along their geometric diagrams.

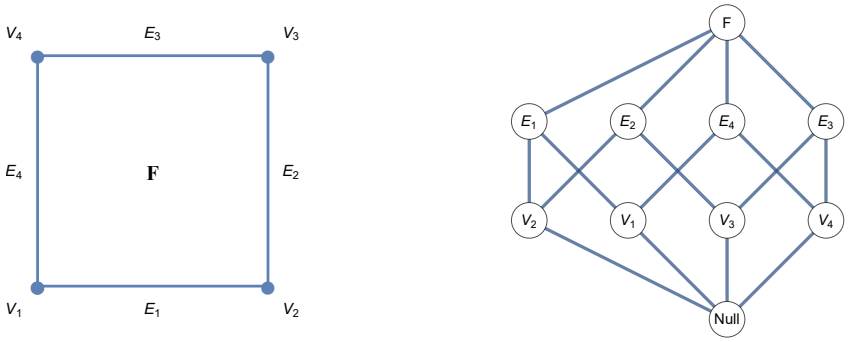
(a) (b)

Figure 4.1: Line. (a). Geometric diagram (b). Hasse diagram



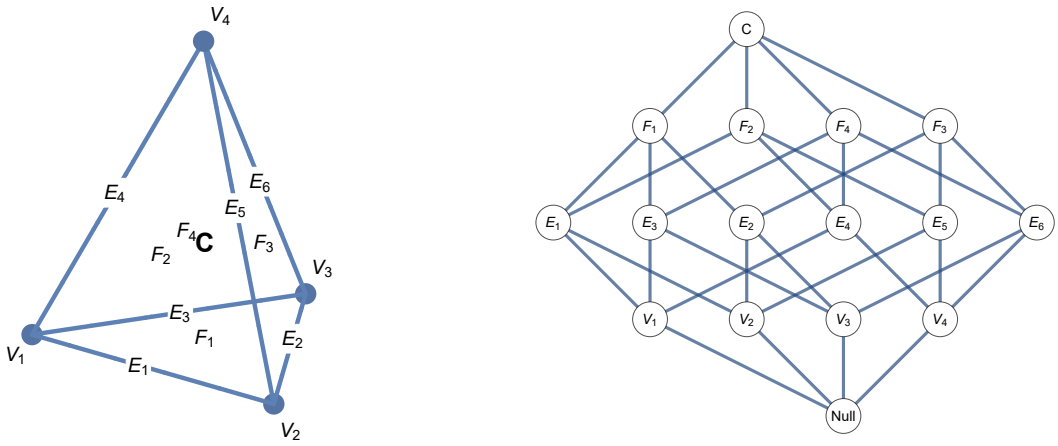
(a) (b)

Figure 4.2: Triangle. (a). Geometric diagram (b). Hasse diagram



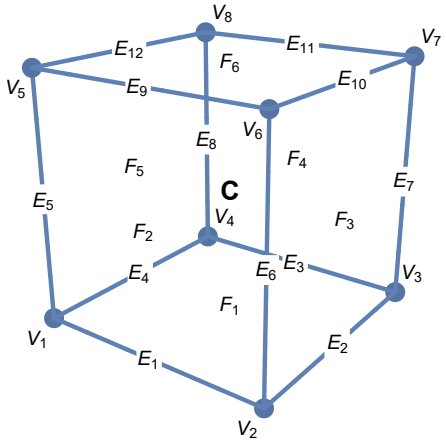
(a) (b)

Figure 4.3: Quadrilateral. (a). Geometric diagramt (b). Hasse diagram

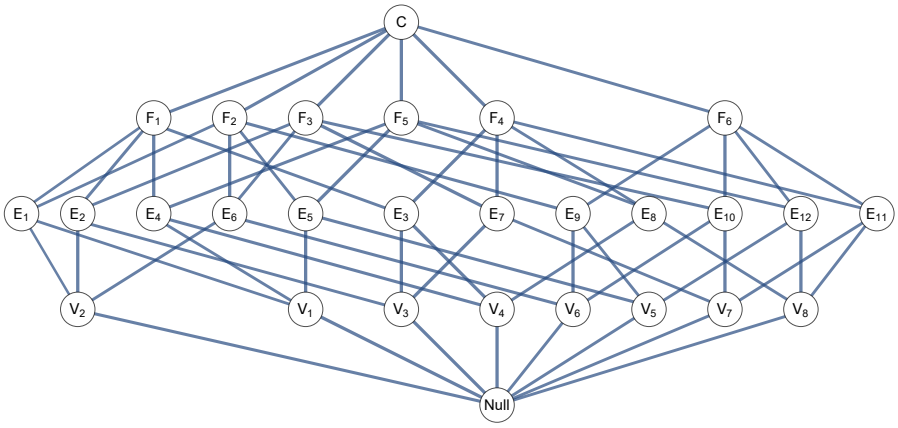


(a) (b)

Figure 4.4: Tetrahedron. (a). Geometric diagram (b). Hasse diagram



(a)



(b)

Figure 4.5: Hexahedron. (a). Geometric diagram (b). Hasse diagram

A Hasse diagram is a type of graph visualization for polytopes and it shows the hierarchy, connectivity, and structure of a polytope. In the geometric diagrams, the vertices are the corners where two edges meet. The vertices coincide on the nodes of linear elements in FEM. Higher order elements have more nodes than their linear versions but their geometry plot remains the same. I emphasized this fact to eliminate confusion between element diagrams in FEM and the geometric diagrams introduced here.

Embedded convex polytope intersection algorithm generates a recursive structure as the result of the intersection between two polytopes. The dimension of the intersection polytope is automatically resolved by the algorithm without prior knowledge. For example, the expected intersection of two triangles in two-dimensional space is a polygon, but in three-dimensional space, it might as well be a line.

4.1.1 Intersection formulation

At the heart of ECPI, there is an equation solver to find intersection points. I formulate the intersection of two shapes directly by equating global coordinates of the shapes. The formulation results in a system of equations and the solution of which describes the intersection between the two shapes. The geometric shape of elements can be described using shape-functions. The shape-functions are fixed parametric functions defined for a reference element in a local coordinate system. Let us formally define vector of shape-functions $\Psi(\mathcal{X})$ in terms of local coordinates vector $\mathcal{X}$ for an n-node element as:

$$\Psi(\mathcal{X}) \triangleq [\ \psi_1(\mathcal{X}) \quad \psi_2(\mathcal{X}) \quad \dots \quad \psi_n(\mathcal{X})\ ]^T \tag{4.1}$$

$$\psi_i(\mathcal{X}) : \mathbb{R}^p \to \mathbb{R} : \mathcal{X} \to \psi_i(\mathcal{X}) \qquad i = 1 \dots n \tag{4.2}$$

Using the shape-functions of each element we construct a mapping between local and global coordinate systems. The global coordinate system is the embedding space and it may be of higher dimension than the elements themselves (e.g. a line embedded in 3D space). This mapping is a core concept in finite element method and is used for creating isoparametric elements. The following formula defines global coordinates of an element in terms of its parametric shape functions and nodal coordinates:

$$x = \sum_{i=1}^{n} \hat{x}(i)\psi_i(\mathcal{X}) = \hat{X}\Psi(\mathcal{X}) \tag{4.3}$$

$$x \triangleq [\ x_1 \quad \dots \quad x_d\ ]^T \tag{4.4}$$

$$\hat{x}(i) \triangleq [\ \hat{x}(i)_1 \quad \dots \quad \hat{x}(i)_d\ ]^T \qquad i = 1 \dots n \tag{4.5}$$

$$\hat{X} = [\ \hat{x}(1) \quad \hat{x}(2) \quad \dots \quad \hat{x}(n)\ ]_{d \times n} \tag{4.6}$$

x is the vector of coordinates in the d-dimensional embedding space and $\hat{x}(i)$ is the nodal coordinate of node i in the global coordinate. From here forward A and B refer to two polytopes we are trying to intersect with each other. We can start by equating the two shapes' coordinates in their embedding space:

$$x_A = \sum_{i=1}^{n_A} \hat{x}_A(i)\psi_{A_i}(\mathcal{X}_A), \qquad x_B = \sum_{j=1}^{n_B} \hat{x}_B(j)\psi_{B_j}(\mathcal{X}_B) \tag{4.7}$$

$$\mathcal{X}_A = \{\mathcal{X}_{A_1}, \dots, \mathcal{X}_{A_p}\} \qquad \mathcal{X}_B = \{\mathcal{X}_{B_1}, \dots, \mathcal{X}_{B_q}\} \tag{4.8}$$

$$f(\mathcal{X}_A, \mathcal{X}_B) \triangleq \sum_{i=1}^{n_A} \hat{x}_A(i)\psi_{A_i}(\mathcal{X}_A) - \sum_{j=1}^{n_B} \hat{x}_B(j)\psi_{B_j}(\mathcal{X}_B) \tag{4.9}$$

$$= \hat{X}_A \Psi_A(\mathcal{X}_A) - \hat{X}_B \Psi_B(\mathcal{X}_B) \tag{4.10}$$

$f(\mathcal{X}_A, \mathcal{X}_B)$ defined above is the vector of coordinate differences between two parametric points on A and B in the embedding space. The l_2 norm of f is the Euclidean distance between the two points. The two shapes A and B have an intersection point for a pair of $\{\mathcal{X}_A, \mathcal{X}_B\}$ that makes the difference vector equal to zero vector. We need to solve the d-dimensional system of equations $f(\mathcal{X}_A, \mathcal{X}_B) = 0$. The system has $dim(\mathcal{X}_A) + dim(\mathcal{X}_B) = p + q$ unknowns. Since a number of equations and parameters do not match in general, we have to use the least-squares method. The least-squares formulation transforms the problem into minimizing the distance between two points on A and B. In order to handle nonlinear equations for higher order elements, we can set up Newton's algorithm for the solver. First we combine all the local shape

parameters and then we proceed by defining the Jacobian of the system with respect to all variables as the following:

$$\mathcal{X} \triangleq \begin{bmatrix} \mathcal{X}_{A_1} & \dots & \mathcal{X}_{A_p} & \mathcal{X}_{B_1} & \dots & \mathcal{X}_{B_q} \end{bmatrix} \tag{4.11}$$

$$f(\mathcal{X}) = f(\mathcal{X}_A, \mathcal{X}_B) \tag{4.12}$$

$$= \hat{X}_A \Psi_A(\mathcal{X}_A) - \hat{X}_B \Psi_B(\mathcal{X}_B) = \begin{bmatrix} \hat{X}_A & \hat{X}_B \end{bmatrix} \cdot \begin{bmatrix} \Psi_A(\mathcal{X}_A) \\ -\Psi_B(\mathcal{X}_B) \end{bmatrix} \tag{4.13}$$

$$J(\mathcal{X}) = \begin{bmatrix} \frac{\partial f(\mathcal{X})}{\partial \mathcal{X}_1} & \dots & \frac{\partial f(\mathcal{X})}{\partial \mathcal{X}_{p+q}} \end{bmatrix}_{d \times (p+q)} \triangleq \hat{X}_{AB} \cdot J_{gen}(\mathcal{X}) \tag{4.14}$$

$$\hat{X}_{AB} \triangleq \begin{bmatrix} \hat{X}_A & \hat{X}_B \end{bmatrix}_{d \times (n_A + n_B)} \tag{4.15}$$

$$J_{gen}(\mathcal{X}) \triangleq \begin{bmatrix} \frac{\partial \Psi_A(\mathcal{X}_A)}{\partial \mathcal{X}_{A_1}} & \dots & \frac{\partial \Psi_A(\mathcal{X}_A)}{\partial \mathcal{X}_{A_p}} & 0 & \dots & 0 \\ 0 & \dots & 0 & -\frac{\partial \Psi_B(\mathcal{X}_B)}{\partial \mathcal{X}_{B_1}} & \dots & -\frac{\partial \Psi_B(\mathcal{X}_B)}{\partial \mathcal{X}_{B_q}} \end{bmatrix}_{(n_A + n_B) \times (p+q)} \tag{4.16}$$

The Jacobian matrix is non-square and not invertible but as mentioned earlier, we can use the least-squares method to solve the system using pseudoinverse. In the case that there is a point intersection we are guaranteed to find the point, otherwise we find two parametric points corresponding to a minimum distance. The iterative update equation for Newton's algorithm is derived below to solve the system $f(\mathcal{X}) = g$. The right-hand side vector g is assumed to be non-zero for reasons which we follow on later.

$$g = f(\mathcal{X}^{i+1}) \approx f(\mathcal{X}^i) + J(\mathcal{X}^0) \cdot (\mathcal{X}^{i+1} - \mathcal{X}^i) \tag{4.17}$$

$$J(\mathcal{X}^i) \cdot \Delta\mathcal{X} \approx g - f(\mathcal{X}^i) \tag{4.18}$$

$$\mathcal{X}^{i+1} = \mathcal{X}^i + \Delta\mathcal{X}, \qquad \Delta\mathcal{X} = (J^T \cdot J)^{-1} \cdot J^T \cdot (g - f(\mathcal{X}^i)) \tag{4.19}$$

After applying the least square method we need to solve a system of $(p+q)$ equations defined by the matrix $S = J^T \cdot J$. If the intersection of two shapes happens to be a point, matrix S is full ranked and has a unique solution. However, if the intersection happens to be more than 0-dimensional, the system becomes rank deficient and has infinite solutions. Each and every solution of a rank deficient matrix still lies on the intersection of two shapes and the solutions form a subspace. A key observation is that the rank deficiency of the matrix S equals the dimension of expected intersection polytope. Rank deficiency(RD) and expected intersection polytope dimension(Dim_X) are defined as:

$$Dim_X = RD = dim(A) + dim(B) - rank(S) \tag{4.20}$$

where $dim(A)$ and $dim(B)$ are manifold dimensions of A and B respectively. The ECPI algorithm adds more equations to the system $f(\mathcal{X}) = 0$ until it obtains a full rank system and solves for point intersections. In the case where the expected intersection is of a dimension higher than zero, the intersection will be a polytope of that dimension. For example in polytopes of dimensions one and two, the points constitute the vertices of edges and the edges become facets of faces. ECPI algorithm anticipates a polytope of a certain dimension and will recursively try to solve for its constituents.

4.1.2 Constraint equations

So far we formulated a system of equations which may be rank deficient and requires adding extra equations to become full rank. We modify our formulation to accommodate for extra constraints $c(\mathcal{X}) = r$.

$$c : \mathbb{R}^{p+q} \to \mathbb{R}^{n_c} : c(\mathcal{X}) = r \tag{4.21}$$

$$f_c(\mathcal{X}) = g \tag{4.22}$$

$$f_c(\mathcal{X}) \triangleq \begin{bmatrix} f(\mathcal{X}) \\ c(\mathcal{X}) \end{bmatrix} \qquad g \triangleq \begin{bmatrix} 0 \\ r \end{bmatrix} \tag{4.23}$$

$$J_c(\mathcal{X}) = \begin{bmatrix} \frac{\partial f(\mathcal{X})}{\partial \mathcal{X}_1} & \cdots & \frac{\partial f(\mathcal{X})}{\partial \mathcal{X}_{p+q}} \\ \frac{\partial c(\mathcal{X})}{\partial \mathcal{X}_1} & \cdots & \frac{\partial c(\mathcal{X})}{\partial \mathcal{X}_{p+q}} \end{bmatrix}_{(d+n_c)\times(p+q)} \tag{4.24}$$

The Newton's method formulation is the same as before once we substitute the new Jacobian matrix J_c.

As hinted earlier the extra constraint equations are used to reduce rank deficiency of the system by recursively limiting the search space of points on the polytopes A and B. The ECPI algorithm requires a set of predefined constraint equations corresponding to facets of the polytopes. The combination of these constraints in compliance with the Hasse diagram of the shape will further limit the parametric space to ridges and peaks of the polytope.

Defining the constraint equations for the finite elements is easily done using isoparametric formulation. The advantage of using isoparametric formulation is that it allows us to reuse the constraint equations of linear elements for all higher order versions as well. In the certain application, we may be interested to know the local coordinate of intersection points for each element. Another advantage of the isoparametric formulation is that since we are solving the equations in terms of local parameters, there is no need to solve a second inverse mapping to find the local coordinates and the forward map to embedding coordinate space is effortless. Table 4.2 lists the constraint equations consistent with facet numbering of figures 4.1 to 4.5 for the basic elements. The complete definition of local coordinates and shape functions for all elements can be found in the appendix. The constraint equations of table 4.2 are

Element	Facet count	Facets	Constraints
Line	2	V_1	$\xi = -1$
		V_2	$\xi = 1$
Triangle	3	E_1	$L_3 = 1 - L_1 - L_2 = 0$
		E_2	$L_1 = 0$
		E_3	$L_2 = 0$
Quadrilateral	4	E_1	$\eta = -1$
		E_2	$\xi = 1$
		E_3	$\eta = 1$
		E_4	$\xi = -1$
Tetrahedron	3	F_1	$L_4 = 1 - L_1 - L_2 - L_3 = 0$
		F_2	$L_3 = 0$
		F_3	$L_1 = 0$
		F_4	$L_2 = 0$
Hexahedron	6	F_1	$\zeta = -1$
		F_2	$\eta = -1$
		F_3	$\xi = 1$
		F_4	$\eta = 1$
		F_5	$\xi = -1$
		F_6	$\zeta = 1$

Table 4.2: Constraint equations of polytope facets

defined in terms of local coordinates and are all commonly linear equations. However being linear is not a requirement. If one decides to define the geometric equations of shapes in terms of embedding coordinates (x,y,z), the constraint equations for high order elements become nonlinear.

4.1.3 ECPI recursive function

The ECPI algorithm loops through the facets of A and B and adds appropriate constraint equations to the system recursively. Using the rank deficiency number RD in each step, the algorithm determines the type of intersection. Algorithm 1 lists a high-level pseudocode for the ECPI algorithm.

Algorithm 1 ECPI high-level psuedocode

```
function ECPI(A, B, S)
    EQs ←assemble system of equations corresponding to state S
    Try solving EQs for an intersection point and calculate the rank
    RD ←rank deficiency of the system
    if a point was found and the point is valid then
        return vertex
    end if
    if system was rank deficient then
        Dim_X ← dimension of expected intersection polytope
        if Dim_X ≠ RD then
            return null
        end if
        for all available constraints of A given S do
            if the constraint is not already satisfied in cHit_A then
                rS ← Prepare state variable and add new constraint to be considered
                PT_A ←ECPI(A, B, rS)                          ▷ 1st recursive call
                if polytope dimension of PT_A = Dim_X − 1 then
                    PT ← Add PT_A to facet of PT if it does not already exist
                    cHit_A ← Update the state of satisfied constraints of A given PT_A
                end if
            end if
        end for
...
```

ECPI function continued ...

for all $facets$ **in** PT **do**

$cHit_B \leftarrow$ Update the state of satisfied constraints of B given the $facet$

end for

for all available constraints of B given S **do**

if the constraint is not already satisfied **then**

$rS \leftarrow$ Prepare state variable and add new constraint to be considered

$PT_B \leftarrow$ECPI(A, B, rS) ▷ 2nd recursive call

if polytope dimension of $PT_B = Dim_X - 1$ **then**

$PT \leftarrow$ Add PT_B to facets of PT if it does not already exist

$cHit_B \leftarrow$ Update the state of satisfied constraints of B given PT_B

end if

end if

end for

end if

validate polytope definition of PT

return PT

end function

The pseudo-code provided in algorithm 1 describes a recursive function realization of the ECPI algorithm. The ECPI function takes two polytopes A and B and a state variable S. We control the logic through the state variable S especially when we make the recursive calls at lines 12 and 25. The state variable S encapsulate all the information necessary for the algorithm to make decisions. The initial ECPI function call will start from a default state S and as the algorithm steps are executed,

the function may decide to make recursive calls. Before making a recursive call, the ECPI function sets a new recursion state rS which will guide the next function call. The state variable includes information about active constraints of A and B , current recursive levels. The recursion level refers to the polytope dimension defined by constrained shapes A and B. For example, if A is a triangle, recursion level of unconstrained A is two. By adding one constraint equation to A, the shape will be constrained to one of its facets which in this case will be an edge of the triangle and this corresponds to recursion level one. The recursion level will stop at zero which corresponds to vertices of a polytope. For the initial function call, we set a default state with no active constraint and recursion levels set to polytope's dimensions of A and B.

The variables $cHit_A$ and $cHit_B$ stand for "constraint hits" and they are used to track which constraints of the shapes are already satisfied. Every time we find an intersection polytope PT_A or PT_B we check to see which constraints they satisfy. The constraint satisfaction checking is also a recursive function itself. To make it more clear, suppose an intersection polytope PT_A is a one dimensional polytope. PT_A will satisfy a constraint equation, only if all of it's points satisfy that constraint. Due to the design of ECPI, we only need to recursively check which constraints are satisfied by all facets of PT_A. If both vertices of the PT_A edge satisfy the same constraints set, we conclude that all points of PT_A also satisfy the same set. The whole procedure of checking "constraint hits" is required to prevent generating redundant intersection polytopes and avoid a computationally expensive cleanup process.

At the last step before the ECPI function returns the result, we verify that output polytope PT has minimum required number of facets.

4.1.4 ECPI example

This section serves to demonstrate the inner workings of the recursive ECPI function through visualizing examples. For the first example we consider the intersection of two lines embedded in $2D$ space and for the second example, we find the intersection between a triangle element and a line element embedded in $2D$ space.

4.1.4.1 Example 1: line-line 2D intersection

Assume we have two arbitrary line elements A and B embedded in $2D$ space. There is a total of 6 test cases to cover all possible scenarios. Figure 4.6 depicts all test cases that will be considered. The expected output will be either a point or a line. The ECPI algorithm steps for each test case is visualized by the intersection

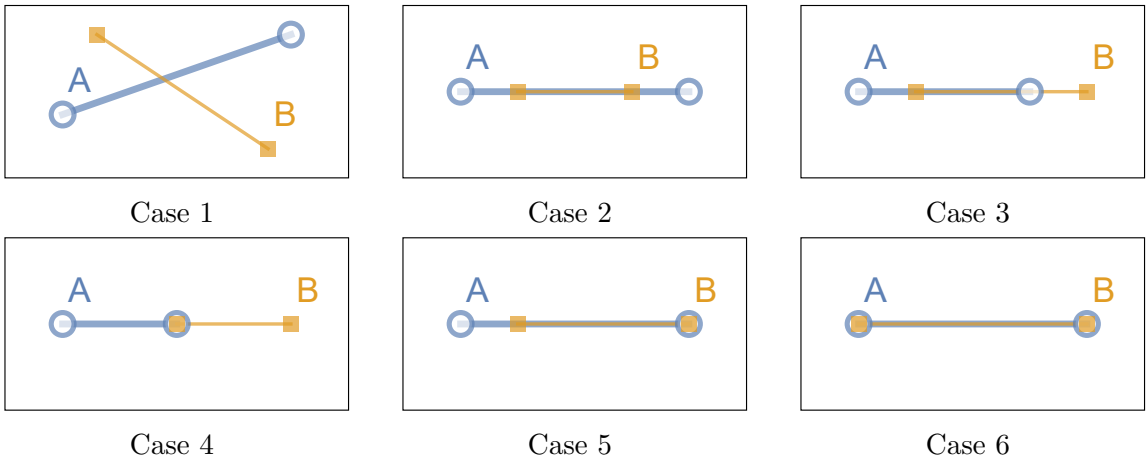
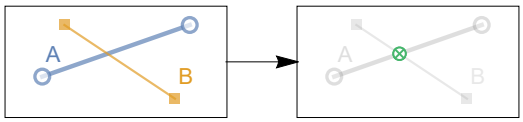


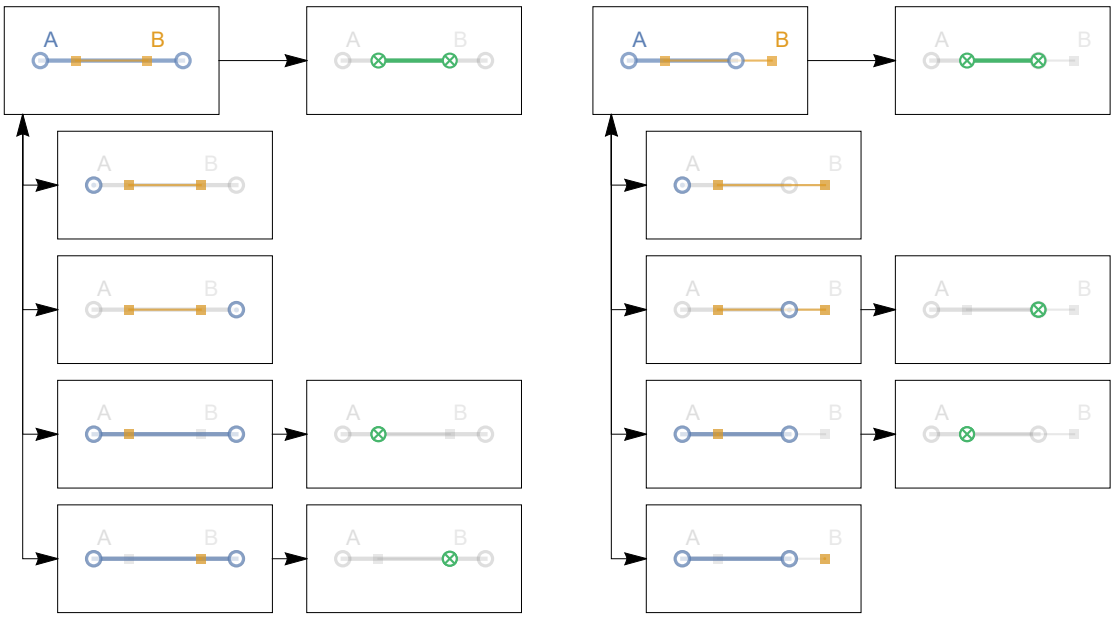
Figure 4.6: $2D$-line-line intersection test cases

sub-problems it tries to solve in each recursive function call. Figure 4.7 visualizes the first test case which is the simple line-line intersection and ECPI finds the intersection point at line 3 of algorithm 1.



Case 1

Figure 4.7: ECPI visualization for $2D$-line-line intersection test cases 1.



Case 2 Case 3

Figure 4.8: ECPI visualization for $2D$-line-line intersection test cases 2 and 3.

Figure 4.7 visualizes the first test case which is the simple line-line intersection and ECPI finds the intersection point at line 3 of algorithm 1. Figures 4.8 and 4.9 visualize the rest of the test cases which involve recursive calls to solve subproblems. All test cases iterate through all vertices of A and B and if the corresponding constraint is not satisfied yet, they will issue a recursive call. Test cases 2 and 3 do not encounter previously satisfied constraints but test cases 4 to 5 will and therefore

make fewer function calls. As mentioned earlier, this mechanism ensures avoiding redundant work and prevents the future cleanup process. In each figure, at the top left corner, we have the problem to be solved and bellow are the recursive function calls. The plots show the constrained search space under given the current state rS. Once all the recursive calls return their results to the main branch, the function generates the output which is placed on the top right corner of each figure.

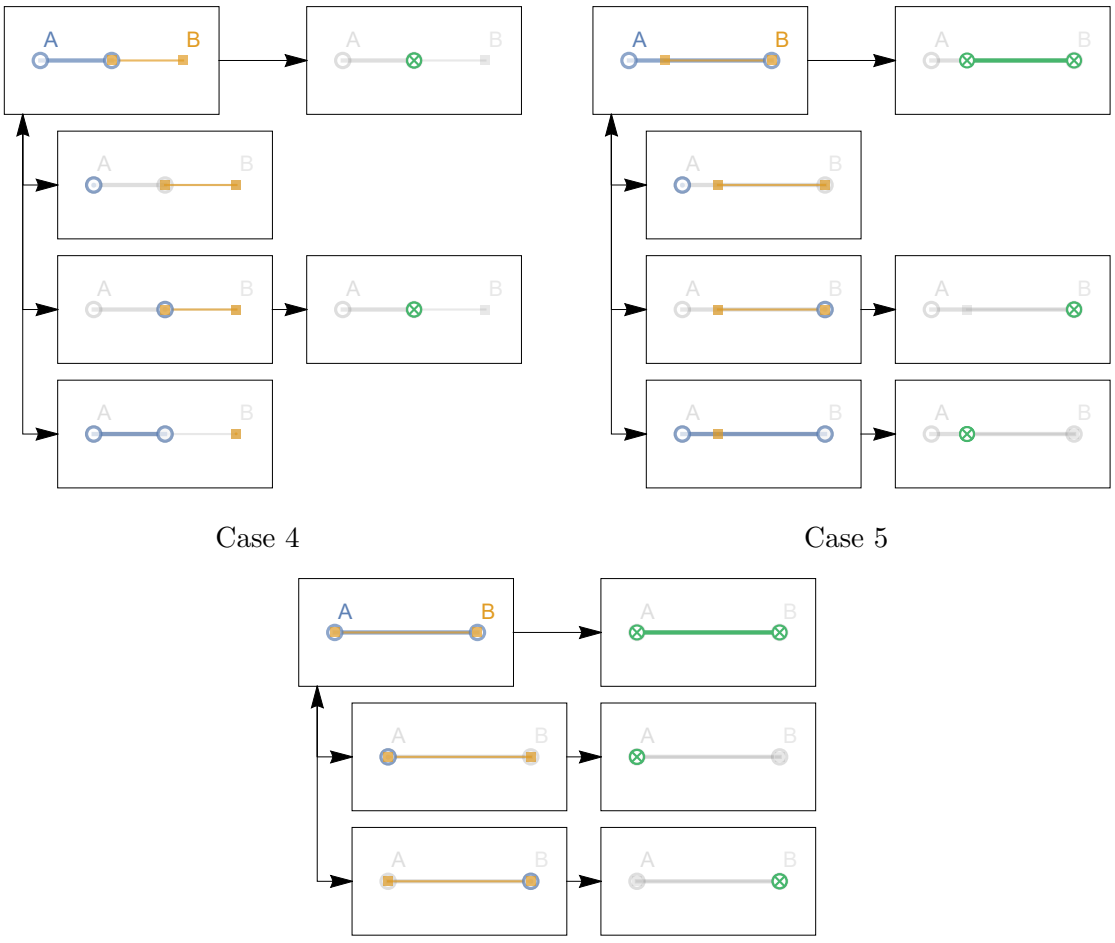
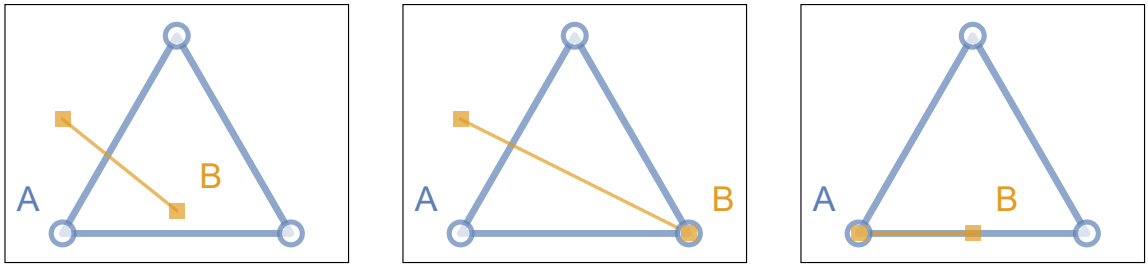


Figure 4.9: ECPI visualization for $2D$-line-line intersection test cases 4,5 and 6.

4.1.4.2 Example 2: line-triangle 2D intersection

In the second example I visualize three test cases depicted in figure 4.10.



Case 1 Case 2 Case 3

Figure 4.10: $2D$-line-triangle intersection test cases

Similar to the previous example, the assumption is we have two input shapes labeled A and B. Figures 4.11 and 4.12 visualize the steps ECPI algorithm takes for each case. The algorithm takes a different path in the recursion execution tree for each test case. The first test case shows all the steps that the algorithm might take in a regular situation.

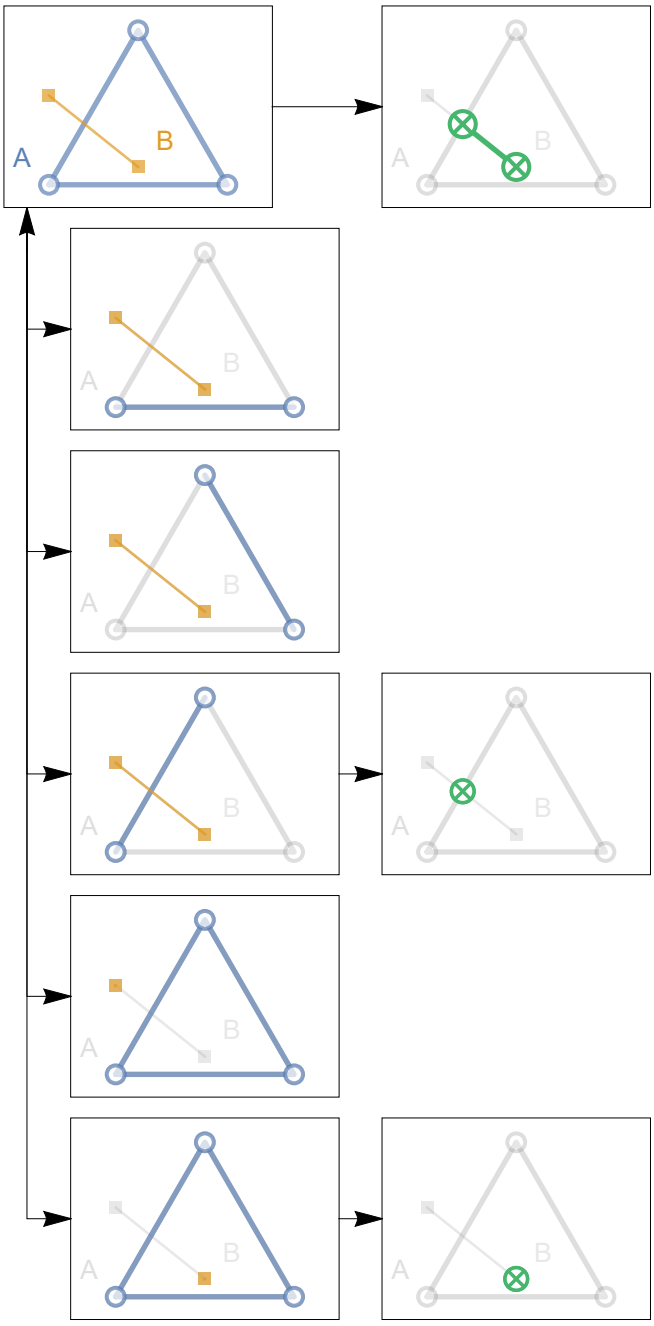
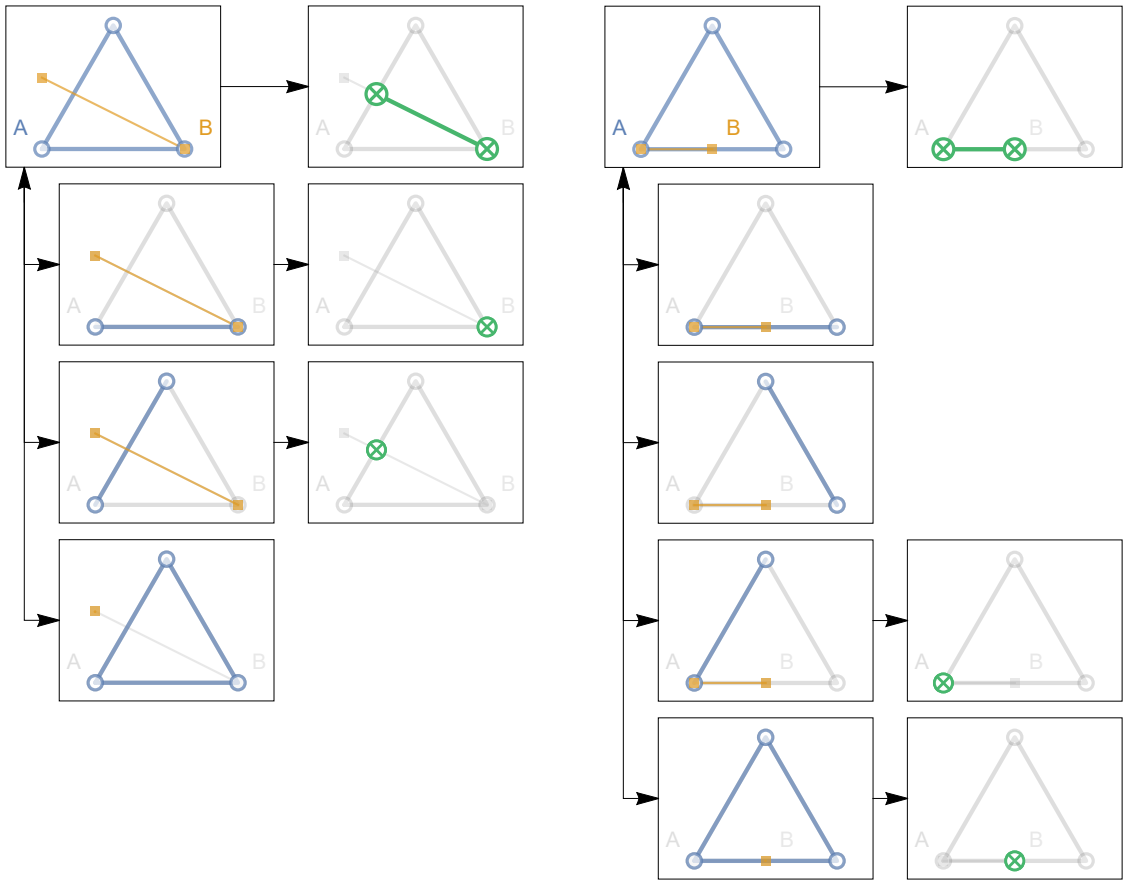


Figure 4.11: ECPI visualization for $2D$-line-triangle intersection test case 1



Case 2 Case 3

Figure 4.12: ECPI visualization for 2D-line-triangle intersection test cases 2 and 3

In the irregular cases, the ECPI algorithm will skip some steps by taking logical decisions depending on the outcome of previously executed steps. For example in the second test case, after the first recursive call returns a point, the algorithm detects that the point is also part of the second constraint of shape A and therefore it will skip over the second constraint when looping through constraints of A. In the same test case, the algorithm also skips the second constraint of B when looping through them since the first point was coinciding with that constraint.

CHAPTER 5

CONCLUTIONS & FUTURE WORKS

Multi-mesh proper orthogonal decomposition problem is solved and is ready for applications which require adaptive mesh refinement for different parameter settings. In this work we used compatible inner products in $\mathbb{F}^n$ for finite dimensional function space. Rigorous mathematical theories have been developed and proved to devise a robust method for constructing reduced order basis in a multi vector space setting. Computation of inner product matrices in finite element analysis requires computation of integrals over common domain of finite elements. A general purpose geometric algorithm is developed which computes intersections of any combination of convex polytopes embedded in any dimension. The embedded convex polytope intersection algorithm can be used as an stand alone tool. All the computer codes are developed in C++ with high standards for portability and extendability. Major achievements in this work are:

- Orthogonal subspace projection theory
- Robust method for finding reduced order basis in a multi-mesh scenario for unstructured grids in all dimensions
- Embedded convex polytope intersection algorithm

For the future works we have a few direction in mind which this research can be extended and applied to. In this work we assumed the final ROB problem is going to be solved on a known target mesh; However the possibility of solving the problem directly on the super-mesh will provide a powerful analysis tool which eliminates the guess work involved in choosing an appropriate target mesh. If the super-mesh

solution is known for the problem we can use it to generate optimal target meshes which are refined in just the right places. The famous mesh adaptation method of [24] which uses Grassman manifold theory can be extended to ROBs defined over super-meshes. The orthogonal subspace projection method combined with ECPI algorithm becomes a powerful tool which can be used in various fields such as multi-physics simulations and image registration. Bellow is a list of future works we are considering for this research:

- Solve the reduced problem directly on the super-mesh
- Use super-mesh solution to estimate optimum mesh density function for the target mesh by fusing snapshots' meshes
- Extend Grassman manifold mesh adaptation [24] technique for the super-mesh reduced order basis method
- Use orthogonal subspace projections in multi-physics simulations
- Develop fast and efficient super-mesh library for structured meshes and local refinements

APPENDIX A

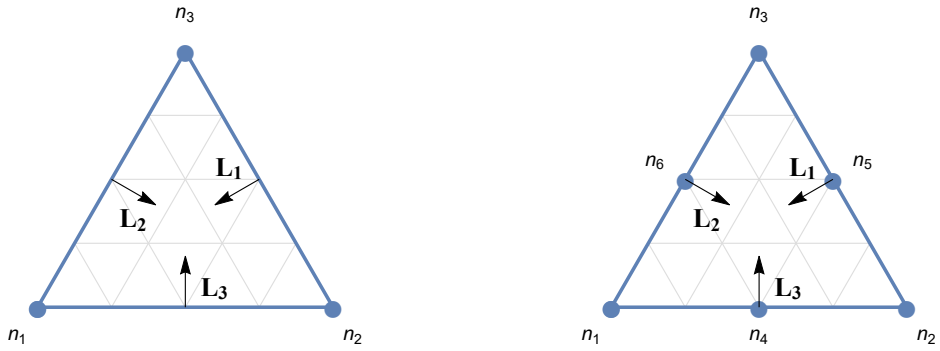
DERIVATIONS & FORMULAE & TEST CASES

(a) (b)

Figure A.1: Line elements node numbering. (a) Linear (b) Quadratic

Element	order	Shape functions
Line	1	$\psi_1(\xi) = (1-\xi)/2$ $\psi_2(\xi) = (1+\xi)/2$
Line	2	$\psi_1(\xi) = \xi(\xi-1)/2$ $\psi_2(\xi) = \xi(\xi+1)/2$ $\psi_3(\xi) = (1-\xi)(1+\xi)/2$

Table A.1: Shape functions of line element



(a) (b)

Figure A.2: Triangle elements node numbering. (a) Linear (b) Quadratic

Element	order	Shape functions
Triangle	1	$\psi_1(L_1, L_2) = L_1$ $\psi_2(L_1, L_2) = L_2$ $\psi_3(L_1, L_2) = 1 - L_1 - L_2$
Triangle	2	$\psi_1(L_1, L_2) = (2L1 - 1)L_1$ $\psi_2(L_1, L_2) = (2L_2 - 1)L_2$ $\psi_3(L_1, L_2) = (L_1 + L_2 - 1)(2L_1 + 2L_2 - 1)$ $\psi_4(L_1, L_2) = 4L_1L_2$ $\psi_5(L_1, L_2) = 4L_2(1 - L_1 - L2)$ $\psi_6(L_1, L_2) = 4(1 - L_1 - L2)L_1$

Table A.2: Shape functions of triangle element

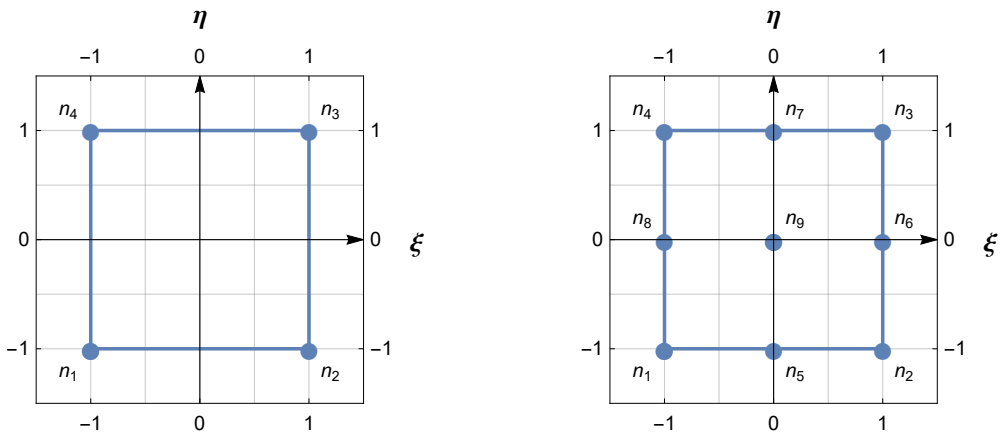
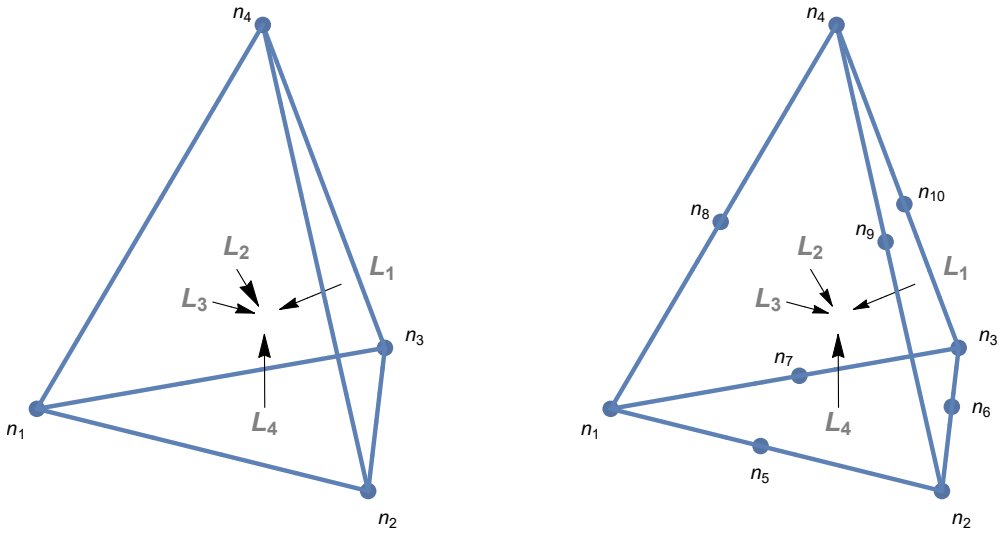


Figure A.3: Quadrilateral elements node numbering. (a) Linear (b) Quadratic

Element	order	Shape functions
Quad	1	$\psi_1(\xi,\eta) = (1-\xi)(1-\eta)/4$ $\psi_2(\xi,\eta) = (1+\xi)(1-\eta)/4$ $\psi_3(\xi,\eta) = (1+\xi)(1+\eta)/4$ $\psi_4(\xi,\eta) = (1-\xi)(1+\eta)/4$
Quad	2	$\psi_1(\xi,\eta) = (-\xi+\xi^2)(-\eta+\eta^2)/4$ $\psi_2(\xi,\eta) = (\xi+\xi^2)(-\eta+\eta^2)/4$ $\psi_3(\xi,\eta) = (\xi+\xi^2)(\eta+\eta^2)/4$ $\psi_4(\xi,\eta) = (-\xi+\xi^2)(\eta+\eta^2)/4$ $\psi_5(\xi,\eta) = (2-2\xi^2)(-\eta+\eta^2)/4$ $\psi_6(\xi,\eta) = (\xi+\xi^2)(2-2\eta^2)/4$ $\psi_7(\xi,\eta) = (2-2\xi^2)(\eta+\eta^2)/4$ $\psi_8(\xi,\eta) = (-\xi+\xi^2)(2-2\eta^2)/4$ $\psi_9(\xi,\eta) = (2-2\xi^2)(2-2\eta^2)/4$

Table A.3: Shape functions of quadrilateral elements



(a) (b)

Figure A.4: Tetrahedral elements node numbering. (a) Linear (b) Quadratic

Element	order	Shape functions
Tet	1	$\psi_1(L_1, L_2, L_3) = L_1$ $\psi_2(L_1, L_2, L_3) = L_2$ $\psi_3(L_1, L_2, L_3) = L_3$ $\psi_4(L_1, L_2, L_3) = 1 - L_1 - L_2 - L_3$
Tet	2	$\psi_1\ (L_1, L_2, L_3) = (2L_1 - 1)L_1$ $\psi_2\ (L_1, L_2, L_3) = (2L_2 - 1)L_2$ $\psi_3\ (L_1, L_2, L_3) = (2L_3 - 1)L_3$ $\psi_4\ (L_1, L_2, L_3) = (1 - 2L_1 - 2L_2 - 2L_3)(1 - L_1 - L_2 - L_3)$ $\psi_5\ (L_1, L_2, L_3) = 4L_1L_2$ $\psi_6\ (L_1, L_2, L_3) = 4L_2L_3$ $\psi_7\ (L_1, L_2, L_3) = 4L_1L_3$ $\psi_8\ (L_1, L_2, L_3) = 4L_1(1 - L_1 - L_2 - L_3)$ $\psi_9\ (L_1, L_2, L_3) = 4L_2(1 - L_1 - L_2 - L_3)$ $\psi_{10}(L_1, L_2, L_3) = 4L_3(1 - L_1 - L_2 - L_3)$

Table A.4: Shape functions of tetrahedral elements

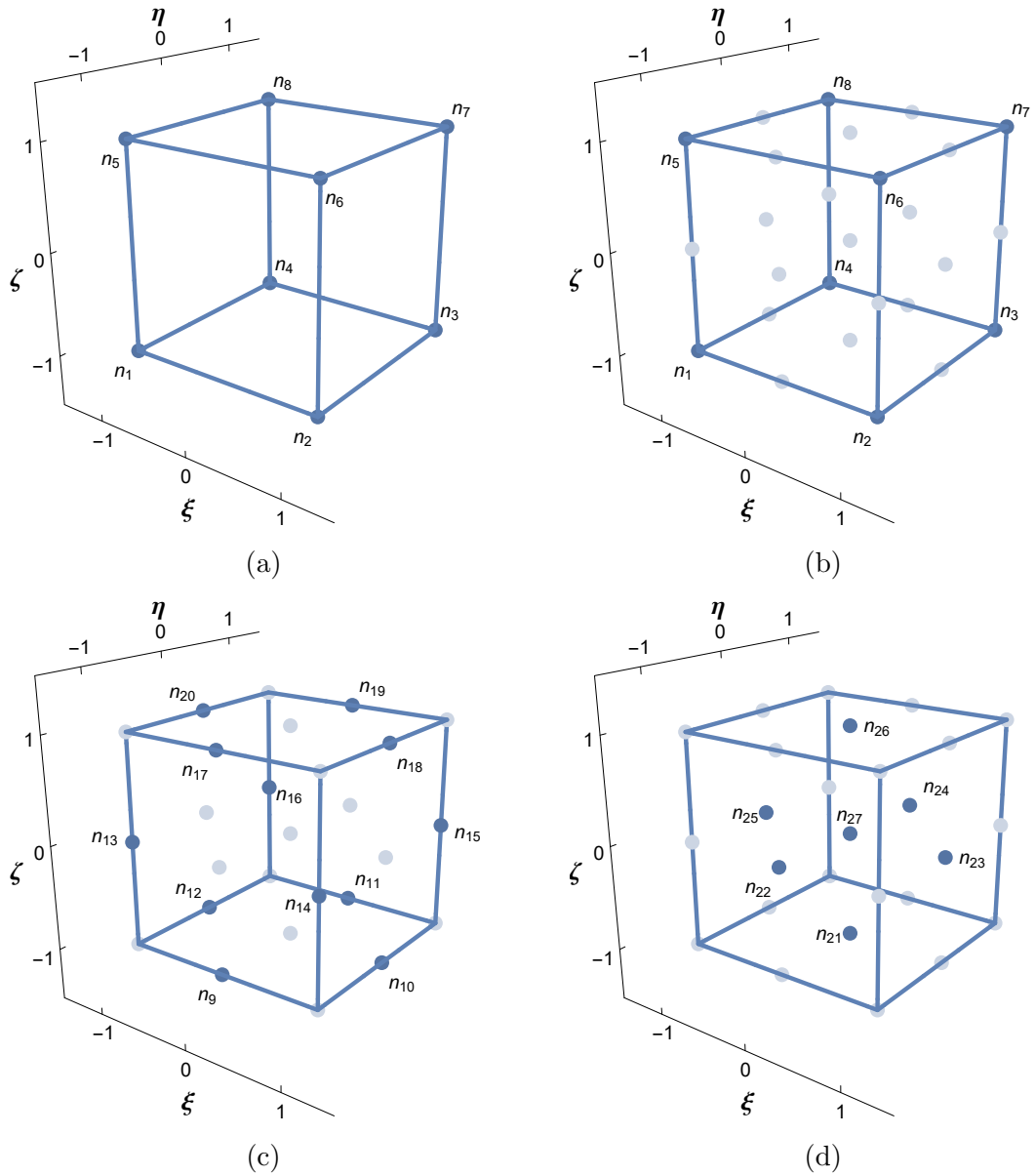


Figure A.5: Hexahedral elements node numbering. (a) Linear (b) Quadratic corner nodes (c) Quadratic mid-edge nodes (d) Quadratic mid-face and mid-cell nodes

Element	order	Shape functions
Hex	1	$\psi_1(\xi,\eta,\zeta) = (1-\xi)(1-\eta)(1-\zeta)/8$
		$\psi_2(\xi,\eta,\zeta) = (1+\xi)(1-\eta)(1-\zeta)/8$
		$\psi_3(\xi,\eta,\zeta) = (1+\xi)(1+\eta)(1-\zeta)/8$
		$\psi_4(\xi,\eta,\zeta) = (1-\xi)(1+\eta)(1-\zeta)/8$
		$\psi_5(\xi,\eta,\zeta) = (1-\xi)(1-\eta)(1+\zeta)/8$
		$\psi_6(\xi,\eta,\zeta) = (1+\xi)(1-\eta)(1+\zeta)/8$
		$\psi_7(\xi,\eta,\zeta) = (1+\xi)(1+\eta)(1+\zeta)/8$
		$\psi_8(\xi,\eta,\zeta) = (1-\xi)(1+\eta)(1+\zeta)/8$
Hex	2	$\psi_1\ (\xi,\eta,\zeta) = -(\zeta-1)\zeta(\eta-1)\eta(\xi-1)\xi/8$
		$\psi_2\ (\xi,\eta,\zeta) = -(\zeta-1)\zeta(\eta-1)\eta\left(\xi^2-1\right)/8$
		$\psi_3\ (\xi,\eta,\zeta) = -(\zeta-1)\zeta\left(\eta^2-1\right)\left(\xi^2-1\right)/8$
		$\psi_4\ (\xi,\eta,\zeta) = -(\zeta-1)\zeta\left(\eta^2-1\right)(\xi-1)\xi/8$
		$\psi_5\ (\xi,\eta,\zeta) = -\left(\zeta^2-1\right)(\eta-1)\eta(\xi-1)\xi/8$
		$\psi_6\ (\xi,\eta,\zeta) = -\left(\zeta^2-1\right)(\eta-1)\eta\left(\xi^2-1\right)/8$
		$\psi_7\ (\xi,\eta,\zeta) = -\left(\zeta^2-1\right)\left(\eta^2-1\right)\left(\xi^2-1\right)/8$
		$\psi_8\ (\xi,\eta,\zeta) = -\left(\zeta^2-1\right)\left(\eta^2-1\right)(\xi-1)\xi/8$
		$\psi_9\ (\xi,\eta,\zeta) = (\zeta-1)\zeta(\eta-1)\eta\xi(\xi+1)/8$
		$\psi_{10}(\xi,\eta,\zeta) = (\zeta-1)\zeta\eta(\eta+1)\left(\xi^2-1\right)/8$
		$\psi_{11}(\xi,\eta,\zeta) = (\zeta-1)\zeta\left(\eta^2-1\right)\xi(\xi+1)/8$
		$\psi_{12}(\xi,\eta,\zeta) = (\zeta-1)\zeta\eta(\eta+1)(\xi-1)\xi/8$
		$\psi_{13}(\xi,\eta,\zeta) = \zeta(\zeta+1)(\eta-1)\eta(\xi-1)\xi/8$
		$\psi_{14}(\xi,\eta,\zeta) = \zeta(\zeta+1)(\eta-1)\eta\left(\xi^2-1\right)/8$
		$\psi_{15}(\xi,\eta,\zeta) = \zeta(\zeta+1)\left(\eta^2-1\right)\left(\xi^2-1\right)/8$
		$\psi_{16}(\xi,\eta,\zeta) = \zeta(\zeta+1)\left(\eta^2-1\right)(\xi-1)\xi/8$
		$\psi_{17}(\xi,\eta,\zeta) = \left(\zeta^2-1\right)(\eta-1)\eta\xi(\xi+1)/8$
		$\psi_{18}(\xi,\eta,\zeta) = \left(\zeta^2-1\right)\eta(\eta+1)\left(\xi^2-1\right)/8$
		$\psi_{19}(\xi,\eta,\zeta) = \left(\zeta^2-1\right)\left(\eta^2-1\right)\xi(\xi+1)/8$
		$\psi_{20}(\xi,\eta,\zeta) = \left(\zeta^2-1\right)\eta(\eta+1)(\xi-1)\xi/8$
		$\psi_{21}(\xi,\eta,\zeta) = -(\zeta-1)\zeta\eta(\eta+1)\xi(\xi+1)/8$
		$\psi_{22}(\xi,\eta,\zeta) = -\zeta(\zeta+1)(\eta-1)\eta\xi(\xi+1)/8$
		$\psi_{23}(\xi,\eta,\zeta) = -\zeta(\zeta+1)\eta(\eta+1)\left(\xi^2-1\right)/8$
		$\psi_{24}(\xi,\eta,\zeta) = -\zeta(\zeta+1)\left(\eta^2-1\right)\xi(\xi+1)/8$
		$\psi_{25}(\xi,\eta,\zeta) = -\zeta(\zeta+1)\eta(\eta+1)(\xi-1)\xi/8$
		$\psi_{26}(\xi,\eta,\zeta) = -\left(\zeta^2-1\right)\eta(\eta+1)\xi(\xi+1)/8$
		$\psi_{27}(\xi,\eta,\zeta) = \zeta(\zeta+1)\eta(\eta+1)\xi(\xi+1)/8$

b

Table A.5: Shape functions of hexahedral elements

A.2 Nullspace calculation

Given an $m \times n$ matrix A, we can calculate basis of the null space using LUQ factorization. L is an $m \times m$ invertible matrix, Q is an $n \times n$ invertible matrix and U is an $m \times n$ upper trapezoidal matrix.

$$A = LUQ \qquad r = Rank(A) \tag{A.1}$$

The $n - r$ columns of Q^{-1} corresponding to the pivotless columns of U are a basis for the null space of A.

$$PA = \tilde{L}\tilde{U} \tag{A.2}$$

$P : m \times m$ permutation matrix

$\tilde{L} : m \times n$ lower trapezoidal matrix with ones on the diagonal

$\tilde{U} : n \times n$ upper triangular matrix write $\tilde{U}$

$$\tilde{U} = \begin{bmatrix} \tilde{U}_{11} & \tilde{U}_{12} \\ 0 & \tilde{U}_{22} \end{bmatrix} \tag{A.3}$$

$\tilde{U}_{11}$: is invertible define L,U and Q as following:

$$L = P^T \begin{bmatrix} \tilde{L} & e_{n+1} & \dots & e_m \end{bmatrix}, \quad U = \begin{bmatrix} \tilde{U}_{11} & 0 \\ 0 & \tilde{U}_{22} \\ 0 & 0 \end{bmatrix}, \quad Q = \begin{bmatrix} I & \tilde{U}_{11}^{-1}\tilde{U}_{12} \\ 0 & I \end{bmatrix} \tag{A.4}$$

A.3 Proofs

Lemma 1. *If V and W are two finite dimensional inner product spaces with dimension n, the mapping f that maps an orthonormal basis of V to an orthonormal basis of W is an isomorphism.*

$$\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots \boldsymbol{a}_n\} : \textit{orthonormal basis of } V \tag{3.2}$$

$$\mathcal{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots \boldsymbol{b}_n\} : \textit{orthonormal basis of } W \tag{3.3}$$

$$f : V \to W : \boldsymbol{v} \in V \to f(\boldsymbol{v}) \in W : f(\boldsymbol{a}_i) = \boldsymbol{b}_i \tag{3.4}$$

Proof. To prove that f is an isomorphism it is enough to show that the inner product is preserved under f. We need to show that the inner product of images of two vectors is the same as inner product of the original vectors.

$$\langle ., . \rangle_V : V \times V \to \mathbb{F}, \qquad \langle ., . \rangle_W : W \times W \to \mathbb{F} \tag{A.5}$$

$$\boldsymbol{v} = v_1\boldsymbol{a}_1 + \ldots + v_n\boldsymbol{a}_n \quad \forall \boldsymbol{v} \in V, \qquad v_i = \langle \boldsymbol{v}, \boldsymbol{a}_i \rangle_V \in \mathbb{F} \quad i = 1, \ldots, n \tag{A.6}$$

$$\boldsymbol{w} = w_1\boldsymbol{b}_1 + \ldots + w_n\boldsymbol{b}_n \quad \forall \boldsymbol{w} \in W, \qquad w_i = \langle \boldsymbol{w}, \boldsymbol{b}_i \rangle_W \in \mathbb{F} \quad i = 1, \ldots, n \tag{A.7}$$

$$\boldsymbol{v}, \boldsymbol{u} \in V, \qquad \boldsymbol{v} = \sum_{i=1}^{n} v_i\boldsymbol{a}_i, \qquad \boldsymbol{u} = \sum_{i=1}^{n} u_i\boldsymbol{a}_i \tag{A.8}$$

$$\langle f(\boldsymbol{v}), f(\boldsymbol{u}) \rangle_W = \langle f(\sum_{j=1}^{n} v_j\boldsymbol{a}_j), f(\sum_{i=1}^{n} u_i\boldsymbol{a}_i) \rangle_W \tag{A.9}$$

$$= \langle \sum_{j=1}^{n} v_j f(\boldsymbol{a}_j), \sum_{i=1}^{n} u_i f(\boldsymbol{a}_i) \rangle_W = \langle \sum_{j=1}^{n} v_j\boldsymbol{b}_j, \sum_{i=1}^{n} u_i\boldsymbol{b}_i \rangle_W \tag{A.10}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} v_j \langle \boldsymbol{b}_j, \boldsymbol{b}_i \rangle_W \bar{u}_i = \sum_{i=1}^{n}\sum_{j=1}^{n} v_j \langle \boldsymbol{a}_j, \boldsymbol{a}_i \rangle_V \bar{u}_i \tag{A.11}$$

$$= \langle \sum_{j=1}^{n} v_j\boldsymbol{a}_j, \sum_{i=1}^{n} u_i\boldsymbol{a}_i \rangle_V = \langle \boldsymbol{v}, \boldsymbol{u} \rangle_V \tag{A.12}$$

□

Lemma 2. *If V is a finite dimensional inner product spaces with dimension n and basis $\Psi = \{\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n\}$, then there exists a modified inner product for $\mathbb{F}^n$ such that V is isomorphic to $\mathbb{F}^n$ under the coordinate map of V to $\mathbb{F}^n$.*

$$\begin{aligned} f :& V \to \mathbb{F}^N : \boldsymbol{\psi}_i \to \boldsymbol{e}_i \quad i = 1, \ldots, n \\ & v \to [v]_\Psi \end{aligned} \tag{3.9}$$

$\boldsymbol{e}_i$ is the standard basis of $\mathbb{F}^n$ and $[\boldsymbol{v}]_\Psi$ is the coordinate of v in basis Ψ. The compatible inner product is defined as:

$$\langle \boldsymbol{v}, \boldsymbol{u} \rangle_V \triangleq \langle [\boldsymbol{v}]_\Psi , G_V [\bar{\boldsymbol{u}}]_\Psi \rangle_{\mathbb{F}^n} \tag{3.10}$$

$$(G_V)_{ij} \triangleq \langle \boldsymbol{\psi}_i, \boldsymbol{\psi}_j \rangle_V \in \mathbb{F}. \tag{3.11}$$

Proof. $\forall \boldsymbol{v}, \boldsymbol{u} \in V$ we have:

$$\boldsymbol{v} = \sum_{i=1}^{n} v_i \boldsymbol{\psi}_i \longmapsto f(\boldsymbol{v}) = \sum_{i=1}^{n} v_i \boldsymbol{e}_i \tag{A.13}$$

$$\boldsymbol{u} = \sum_{i=1}^{n} u_i \boldsymbol{\psi}_i \longmapsto f(\boldsymbol{u}) = \sum_{i=1}^{n} u_i \boldsymbol{e}_i \tag{A.14}$$

$$[\boldsymbol{v}]_\Psi = \{v_i \in \mathbb{F} \ : \ i = 1 \ldots n\} \tag{A.15}$$

$$[\boldsymbol{u}]_\Psi = \{u_i \in \mathbb{F} \ : \ i = 1 \ldots n\} \tag{A.16}$$

We start with the abstract definition of the inner product in vector space V. We assume the general case for the field $\mathbb{F}$ where it can be replaced by $\mathbb{C}$ or $\mathbb{R}$.

$$\langle ., . \rangle_{\mathbb{F}^n} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F} \qquad \langle ., . \rangle_V : V \times V \to \mathbb{F} \tag{A.17}$$

$$\langle \boldsymbol{v}, \boldsymbol{u} \rangle_V = \langle \sum_{i=1}^{n} v_i \boldsymbol{\psi}_i, \sum_{j=1}^{n} u_j \boldsymbol{\psi}_j \rangle_V \tag{A.18}$$

$$= \sum_{i=1}^{n} v_i \langle \boldsymbol{\psi}_i, \sum_{j=1}^{n} u_j \boldsymbol{\psi}_j \rangle_V = \sum_{i=1}^{n} \sum_{j=1}^{n} v_i \langle \boldsymbol{\psi}_i, \boldsymbol{\psi}_j \rangle_V \bar{u}_j = \sum_{i=1}^{n} v_i \sum_{j=1}^{n} g_{ij} \bar{u}_j$$

Since $\boldsymbol{e}_i$'s are orthonormal we can add another summation and the term $\langle \boldsymbol{e}_i, \boldsymbol{e}_k \rangle$ to the right hand side of the equation to get:

$$\langle \boldsymbol{v}, \boldsymbol{u} \rangle_V = \sum_{k=1}^{n} \sum_{i=1}^{n} v_i \langle \boldsymbol{e}_i, \boldsymbol{e}_k \rangle \sum_{j=1}^{n} (G_V)_{ij} \bar{u}_j \tag{A.19}$$

We can further switch the index i to k in the term $(G_V)_{ij}$ to get:

$$\begin{aligned} \langle \boldsymbol{v}, \boldsymbol{u} \rangle_V &= \sum_{k=1}^{n} \sum_{i=1}^{n} v_i \langle \boldsymbol{e}_i, \boldsymbol{e}_k \rangle \sum_{j=1}^{n} (G_V)_{kj} \bar{u}_j \qquad \text{(A.20)} \\ &= \langle \sum_{i=1}^{n} v_i \boldsymbol{e}_i, \sum_{k=1}^{n} \sum_{j=1}^{n} (G_V)_{kj} \bar{u}_j \boldsymbol{e}_k \rangle \\ &= \langle \sum_{i=1}^{n} v_i \boldsymbol{e}_i, \sum_{k=1}^{n} (G_V \left[\bar{\boldsymbol{u}} \right]_\Psi)_k \boldsymbol{e}_k \rangle \\ &= \langle \left[\boldsymbol{v} \right]_\Psi, G_V \left[\bar{\boldsymbol{u}} \right]_\Psi \rangle_{\mathbb{F}^n} \qquad \text{(A.21)} \end{aligned}$$

□

Theorem 1. *Suppose V is an infinite dimensional function space with inner product $\langle ., . \rangle_V$. If A and B are finite dimensional subspaces of V with corresponding basis sets $\mathcal{A}$ and $\mathcal{B}$, there exist a projection transformation $P_{\mathcal{BA}}$, which projects vectors from A to B with respect to basis sets $\mathcal{A}$ and $\mathcal{B}$.*

$$A, B \subset V, \qquad dim(A) = m, \qquad dim(B) = n$$

$$\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m\} : a\ basis\ of\ A \tag{3.12}$$

$$\mathcal{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\} : a\ basis\ of\ B \tag{3.13}$$

$$P_{\mathcal{BA}} : A \to B : \boldsymbol{v}_A \in A \to P_{\mathcal{BA}}[\boldsymbol{v}_A]_{\mathcal{A}} \in B \tag{3.14}$$

$$P_{\mathcal{BA}} = G_{\mathcal{BB}}^{-1} G_{\mathcal{BA}} \tag{3.15}$$

$$(G_{\mathcal{BB}})_{ij} \triangleq \overline{\langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle}_V = \langle \boldsymbol{b}_j, \boldsymbol{b}_i \rangle_V \in \mathbb{F}, \qquad \boldsymbol{b}_i, \boldsymbol{b}_j \in \mathcal{B}, \quad i, j = 1 \ldots n \tag{3.16}$$

$$(G_{\mathcal{BA}})_{ij} \triangleq \overline{\langle \boldsymbol{b}_i, \boldsymbol{a}_j \rangle}_V = \langle \boldsymbol{a}_j, \boldsymbol{b}_i \rangle_V \in \mathbb{F}, \qquad \boldsymbol{b}_i \in \mathcal{B}, \quad i = 1 \ldots n \tag{3.17}$$

$$\boldsymbol{a}_j \in \mathcal{A}, \quad j = 1 \ldots m$$

Proof. First decompose V into the direct sum of B and $B^{\perp}$. Any vector $\boldsymbol{v} \in V$ can be uniquely decomposed into it's projections onto B and $B^{\perp}$.

$$V = B \oplus B^{\perp} \tag{A.22}$$

$$\boldsymbol{v}_A = \boldsymbol{v}_B + \boldsymbol{v}_{B^{\perp}}, \qquad \boldsymbol{v}_A \in A \subset V, \quad \boldsymbol{v}_B \in B, \quad \boldsymbol{v}_{B^{\perp}} \in B^{\perp} \tag{A.23}$$

we can represent $\boldsymbol{v}_B$ and $\boldsymbol{v}$ in terms of basis sets $\mathcal{A}$ and $\mathcal{B}$ using the following notation:

$$[\boldsymbol{v}_B]_{\mathcal{B}} = \{v_{Bi} \in \mathbb{F} \ : \ i = 1 \ldots n\}, \qquad \boldsymbol{v}_B = \sum_{i=1}^{n} v_{Bi} \boldsymbol{b}_i \tag{A.24}$$

$$[\boldsymbol{v}_A]_{\mathcal{A}} = \{v_{Ai} \in \mathbb{F} \ : \ i = 1 \ldots m\}, \qquad \boldsymbol{v}_A = \sum_{i=1}^{m} v_{Ai} \boldsymbol{a}_i \tag{A.25}$$

Substitute definition of $\boldsymbol{v}_B$ and $\boldsymbol{v}$ into the decomposition equation to get:

$$\sum_{j=1}^{n} v_{Bj}\boldsymbol{b}_j + \boldsymbol{v}_{B^\perp} = \sum_{k=1}^{m} v_{Ak}\boldsymbol{a}_k \tag{A.26}$$

apply inner product to both sides of the equation n times to get a system of equations with coordinates v_{Bj} as the unknowns:

$$\langle\sum_{j=1}^{n} v_{Bj}\boldsymbol{b}_j, \boldsymbol{b}_i\rangle_V + \langle\boldsymbol{v}_{B^\perp}, \boldsymbol{b}_i\rangle_V = \langle\sum_{k=1}^{m} v_{Ak}\boldsymbol{a}_k, \boldsymbol{b}_k\rangle_V, \qquad i = 1\ldots n \tag{A.27}$$

$$\langle\boldsymbol{b}_i, \boldsymbol{v}_{B^\perp}\rangle_V = 0, \qquad \forall\boldsymbol{b}_i \in \mathcal{B} \tag{A.28}$$

$$\sum_{j=1}^{n} v_{Bj}\langle\boldsymbol{b}_j, \boldsymbol{b}_i\rangle_V = \sum_{k=1}^{m} v_{Ak}\langle\boldsymbol{a}_k, \boldsymbol{b}_i\rangle_V, \qquad i = 1\ldots n \tag{A.29}$$

Form a system of equations using all n equations above and simply define $G_{\mathcal{BB}}$ and $G_{\mathcal{BA}}$ as following and the proof will be complete:

$$(G_{\mathcal{BB}})_{ij} \triangleq \langle\boldsymbol{b}_j, \boldsymbol{b}_i\rangle_V, \qquad i, j = 1\ldots n \tag{A.30}$$

$$(G_{\mathcal{BA}})_{ij} \triangleq \langle\boldsymbol{a}_j, \boldsymbol{b}_i\rangle_V, \qquad i = 1\ldots n, \quad j = 1\ldots m \tag{A.31}$$

$$\sum_{j=1}^{n} (G_{\mathcal{BB}})_{ij} v_{B_j} = \sum_{k=1}^{m} (G_{\mathcal{BA}})_{ik} v_{Ak}, \qquad i = 1\ldots n \tag{A.32}$$

$$G_{\mathcal{BB}}[\boldsymbol{v}_B]_{\mathcal{B}} = G_{\mathcal{BA}}[\boldsymbol{v}_A]_{\mathcal{A}} \tag{A.33}$$

□

Proposition 1. *Suppose V is an infinite dimensional function space with inner product $\langle ., . \rangle_V$, and A and B are finite dimensional subspaces of V with corresponding basis sets $\mathcal{A}$ and $\mathcal{B}$. If the vector $\boldsymbol{v}_A \in A$ is the projection of $\boldsymbol{v}$, then the true projection of $\boldsymbol{v}$ onto B is attainable through the subspace projection defined in theorem 1 only when $\boldsymbol{v} - \boldsymbol{v}_A$ has no component on B.*

Proof.

$$A, B \subset V, \qquad dim(A) = m, \qquad dim(B) = n$$

$$\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m\} : \text{a basis of } A \tag{A.34}$$

$$\mathcal{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\} : \text{a basis of } B \tag{A.35}$$

First decompose V into the direct sum of A and $A^\perp$. Any vector $\boldsymbol{v} \in V$ is uniquely decomposed into it's projections onto A and $A^\perp$.

$$V = A \oplus A^\perp, \qquad \boldsymbol{v} = \boldsymbol{v}_A + \boldsymbol{v}_{A^\perp}, \qquad \boldsymbol{v}_A \in A, \quad \boldsymbol{v}_{A^\perp} \in A^\perp \tag{A.36}$$

$$[\boldsymbol{v}_A]_\mathcal{A} = \{v_{Ai} \in \mathbb{F} \ : \ i = 1 \ldots m\}, \qquad \boldsymbol{v}_A = \sum_{i=1}^{m} v_{Ai} \boldsymbol{a}_i \tag{A.37}$$

We will do the same procedure for B as well.

$$V = B \oplus B^\perp, \qquad \boldsymbol{v} = \boldsymbol{v}_B + \boldsymbol{v}_{B^\perp}, \qquad \boldsymbol{v}_B \in B, \quad \boldsymbol{v}_{B^\perp} \in B^\perp \tag{A.38}$$

$$[\boldsymbol{v}_B]_\mathcal{B} = \{v_{Bi} \in \mathbb{F} \ : \ i = 1 \ldots n\}, \qquad \boldsymbol{v}_B = \sum_{i=1}^{n} v_{Bi} \boldsymbol{b}_i \tag{A.39}$$

Next we equate the two decomposition of $\boldsymbol{v}$ to get:

$$\boldsymbol{v}_B + \boldsymbol{v}_{B^\perp} = \boldsymbol{v}_A + \boldsymbol{v}_{A^\perp} \tag{A.40}$$

Using the direct sum definition of $V = B \oplus B^\perp$, we can decompose the vector $V_{A^\perp}$ into components on B and $B^\perp$:

$$\boldsymbol{v}_{A^\perp} = \boldsymbol{v}_{A^\perp_B} + \boldsymbol{v}_{A^\perp_{B^\perp}} \tag{A.41}$$

$$[\boldsymbol{v}_{A_B^\perp}]_{\mathcal{B}} = \left\{ v_{A_B^\perp i} \in \mathbb{F} \ : \ i = 1 \dots n \right\}, \qquad \boldsymbol{v}_{A_B^\perp} = \sum_{i=1}^{n} v_{A_B^\perp i} \boldsymbol{b}_i \tag{A.42}$$

Substitute decomposition of $\boldsymbol{v}_{A^\perp}$ into equation to get:

$$\boldsymbol{v}_B + \boldsymbol{v}_{B^\perp} = \boldsymbol{v}_A + \boldsymbol{v}_{A_B^\perp} + \boldsymbol{v}_{A_{B^\perp}^\perp} \tag{A.43}$$

apply inner product with respect to basis vector of B to the both sides of the equation:

$$\langle \boldsymbol{v}_B, \boldsymbol{b}_i \rangle_V = \langle \boldsymbol{v}_A, \boldsymbol{b}_i \rangle_V + \langle \boldsymbol{v}_{A_B^\perp}, \boldsymbol{b}_i \rangle_V \qquad i = 1 \dots n \tag{A.44}$$

$$\langle \sum_{j=1}^{n} v_{Bj} \boldsymbol{b}_j, \boldsymbol{b}_i \rangle_V = \langle \sum_{k=1}^{m} v_{Ak} \boldsymbol{a}_k, \boldsymbol{b}_i \rangle_V + \langle \sum_{l=1}^{n} v_{A_B^\perp l} \boldsymbol{b}_l, \boldsymbol{b}_i \rangle_V \qquad i = 1 \dots n \tag{A.45}$$

$$\sum_{j=1}^{n} v_{Bj} \langle \boldsymbol{b}_j, \boldsymbol{b}_i \rangle_V = \sum_{k=1}^{m} v_{Ak} \langle \boldsymbol{a}_k, \boldsymbol{b}_i \rangle_V + \sum_{l=1}^{n} v_{A_B^\perp l} \langle \boldsymbol{b}_l, \boldsymbol{b}_i \rangle_V \qquad i = 1 \dots n \tag{A.46}$$

From theorem 1 substitute the inner products above with definitions of $G_{\mathcal{BB}}$ and $G_{\mathcal{BA}}$ to get:

$$\sum_{j=1}^{n} v_{Bj} (G_{\mathcal{BB}})_{ij} = \sum_{k=1}^{m} v_{Ak} (G_{\mathcal{BA}})_{ik} + \sum_{l=1}^{n} v_{A^\perp Bl} (G_{\mathcal{BB}})_{il} \qquad i = 1 \dots n \tag{A.47}$$

Next we rewrite the system of equations in matrix form with v_{Bi} as the unknowns:

$$G_{\mathcal{BB}} [\boldsymbol{v}_B]_{\mathcal{B}} = G_{\mathcal{BA}} [\boldsymbol{v}_A]_{\mathcal{A}} + G_{\mathcal{BB}} [\boldsymbol{v}_{A_B^\perp}]_{\mathcal{B}} \tag{A.48}$$

The coordinate $[\boldsymbol{v}_B]_{\mathcal{B}}$ can be found by solving the system above:

$$[\boldsymbol{v}_B]_{\mathcal{B}} = G_{\mathcal{BB}}^{-1} G_{\mathcal{BA}} [\boldsymbol{v}_A]_{\mathcal{A}} + [\boldsymbol{v}_{A_B^\perp}]_{\mathcal{B}} = P_{\mathcal{BA}} [\boldsymbol{v}_A]_{\mathcal{A}} + [\boldsymbol{v}_{A_B^\perp}]_{\mathcal{B}} \tag{A.49}$$

We have shown that the true projection of the vector $\boldsymbol{v}$ onto subspace B equals the subspace projection of $\boldsymbol{v}_A$ only when the component of $\boldsymbol{v}$ in the complement subspace of A does not have any component in B or in other terms:

$$[\boldsymbol{v}_B]_{\mathcal{B}} = P_{\mathcal{BA}} [\boldsymbol{v}_A]_{\mathcal{A}} \iff [\boldsymbol{v}_{A_B^\perp}]_{\mathcal{B}} = 0 \tag{A.50}$$

□

A.4 ECPI test cases

Here I present limited test cases for different combination of polytopes provided as inputs to ECPI algorithm. Table A.6 lists available test cases. In each test cases two shapes are given as inputs and the intersection is the output of the algorithm. In the figures, The red and blue shapes are the inputs and the green shapes are the intersections. Be aware that the green shape may overlay on top of other shapes and block their visualization. I have tried to cover as many degenerate cases as possible for lower dimensional test cases, but for higher dimensions, due to combinatorial nature of the test cases, I only provided a few key tests cases.

Embedding dimension	Shape A	Shape B	Figure number
2D	Line	Line	A.6
2D	Line	Triangle	A.7
2D	Triangle	Triangle	A.8
2D	Triangle	Quadrilateral	A.9
3D	Tetrahedron	Tetrahedron	A.10
3D	Hexahedron	Tetrahedron	A.11
3D	Line	Triangle	A.12
3D	Quadrilateral	Tetrahedron	A.13

Table A.6: Test cases guide

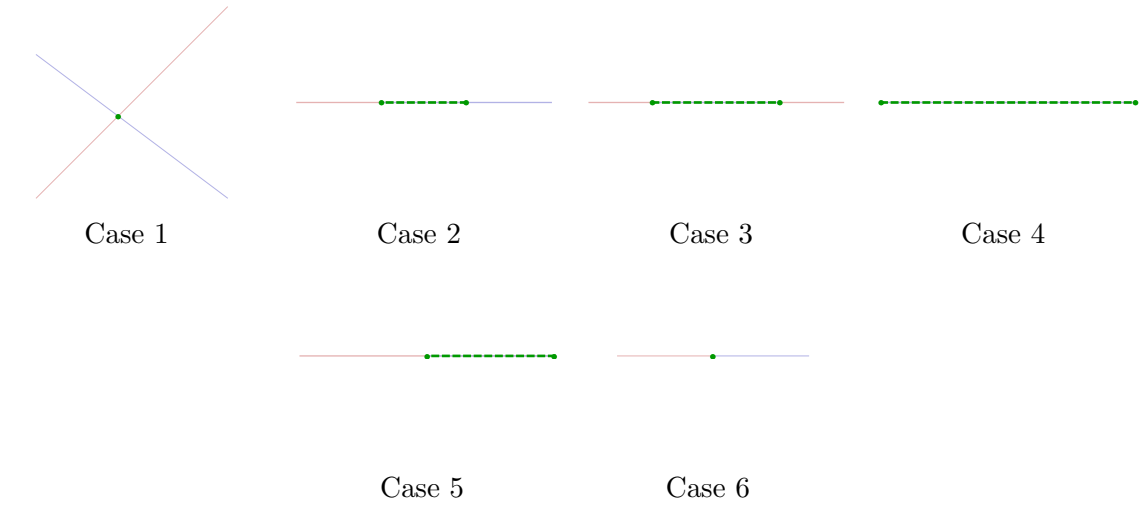


Figure A.6: 2D line-line intersection

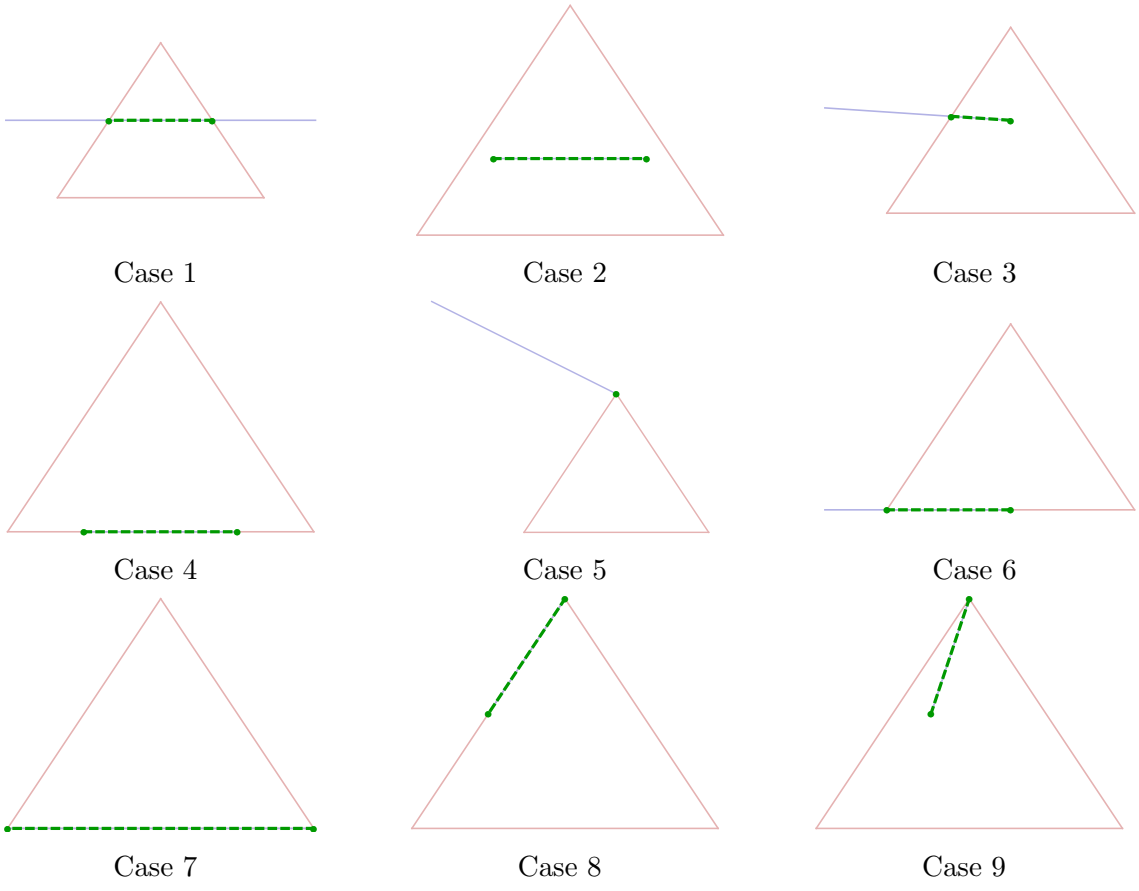


Figure A.7: 2D line-triangle intersection

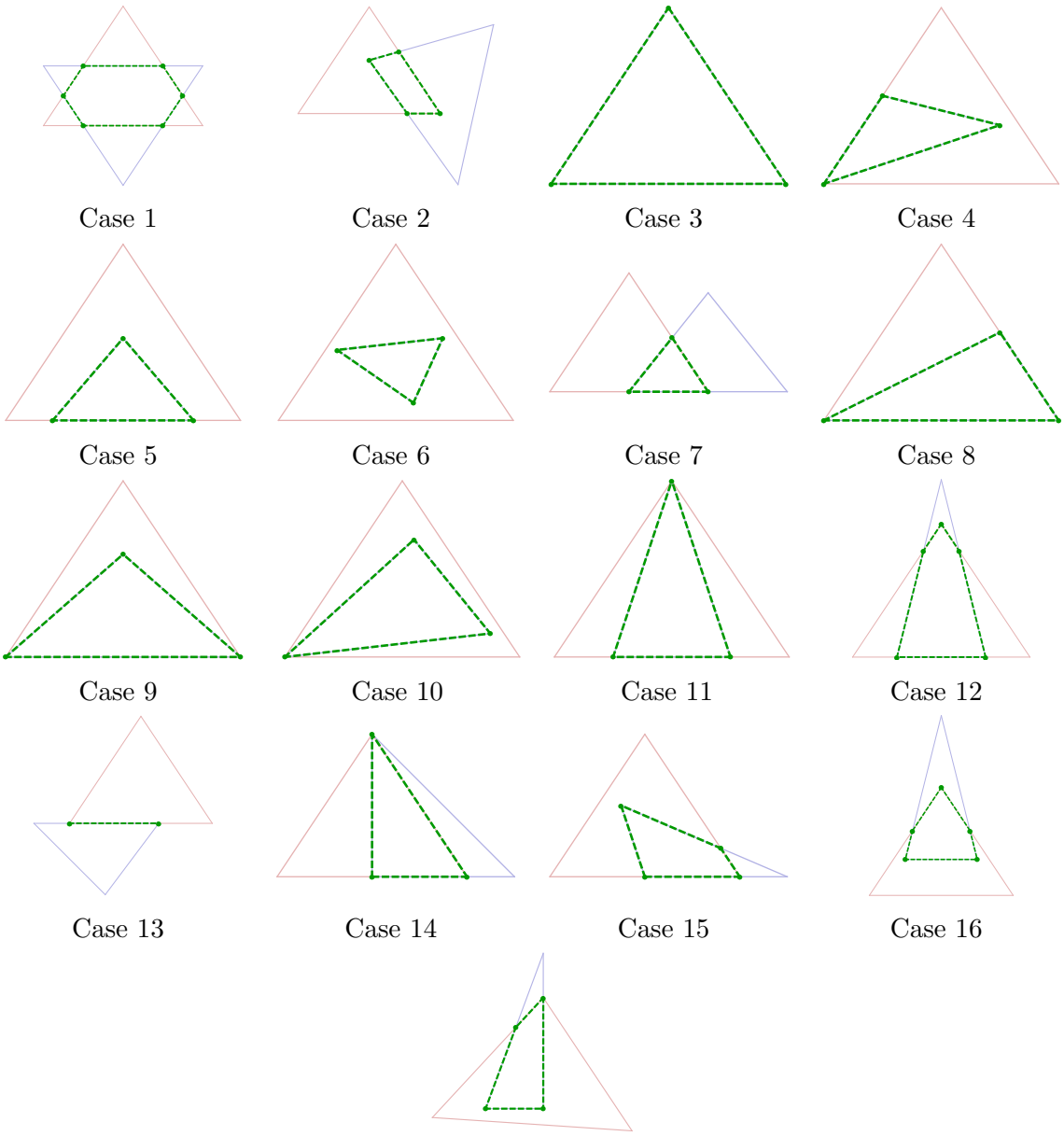


Figure A.8: 2D triangle-triangle intersection

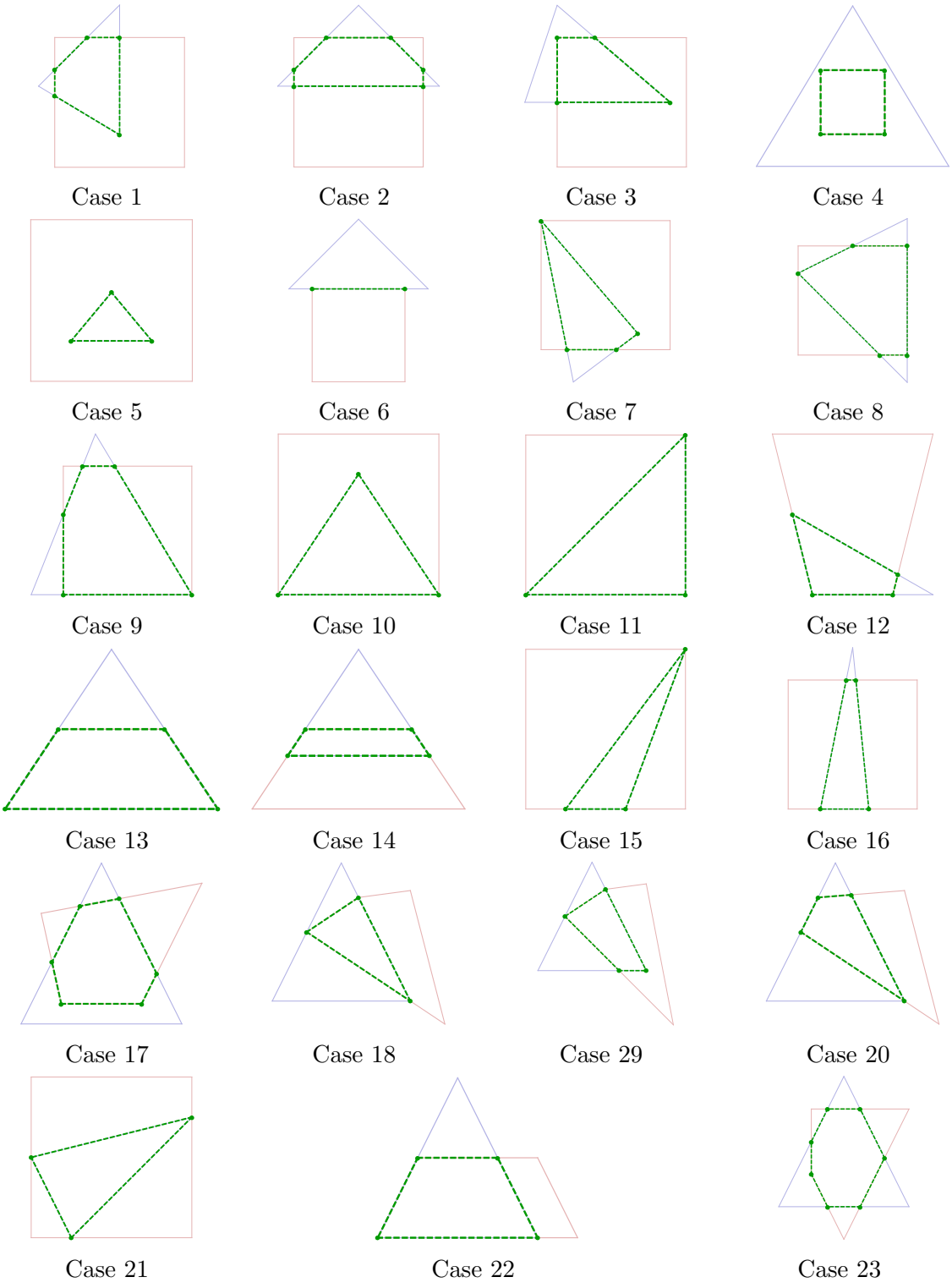
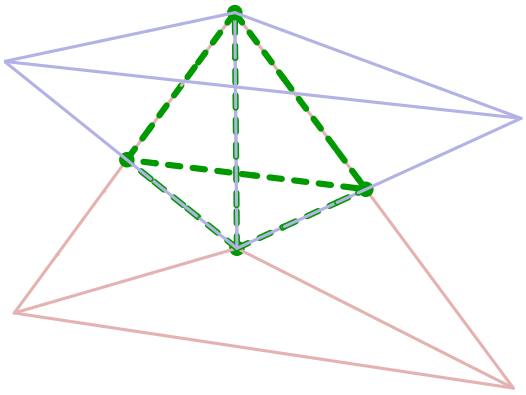
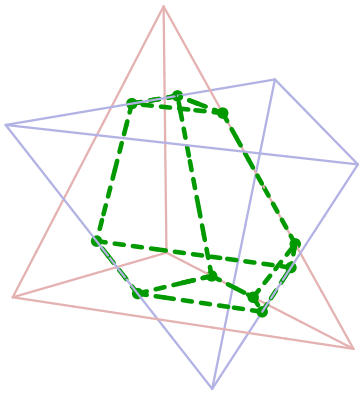


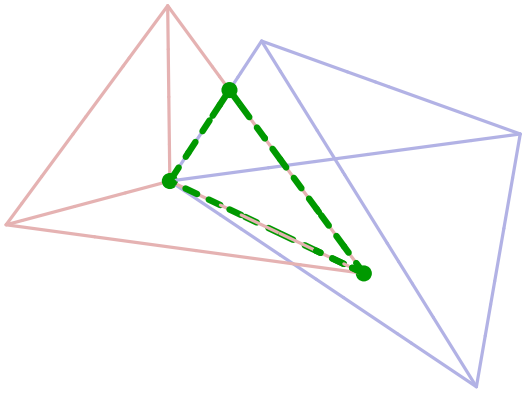
Figure A.9: 2D triangle-Quadrilateral intersection



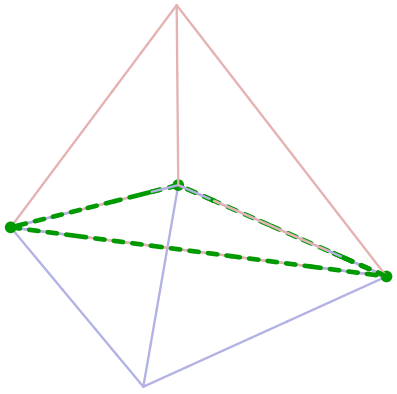
Case 1



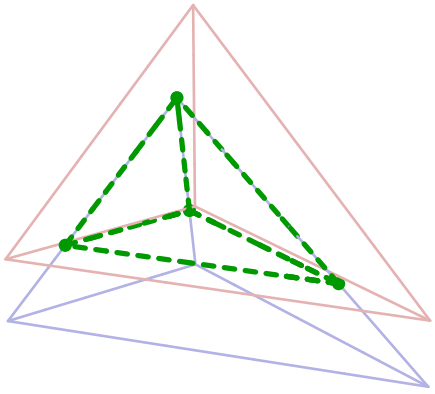
Case 2



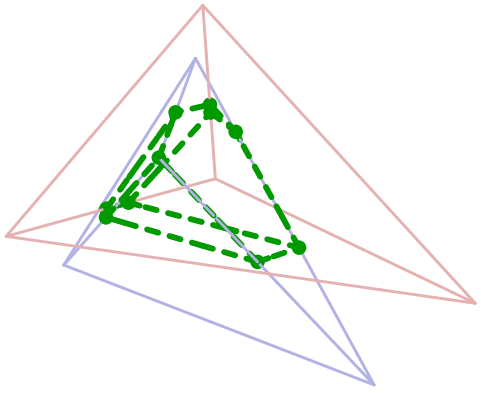
Case 3



Case 4

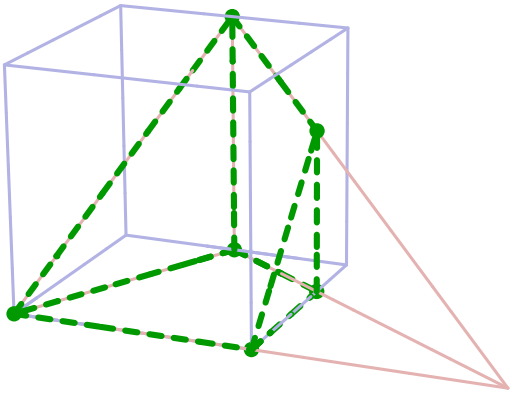


Case 5

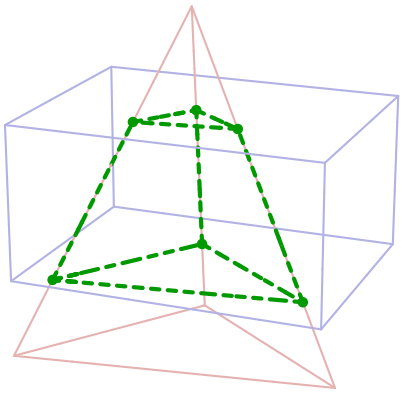


Case 6

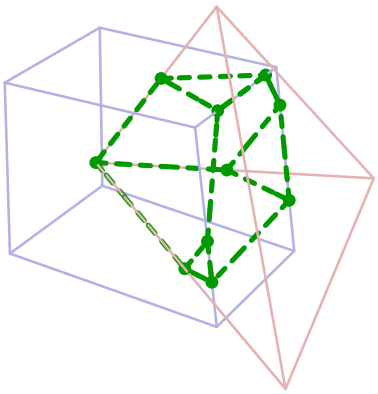
Figure A.10: 3D Tetrahedron-Tetrahedron intersection



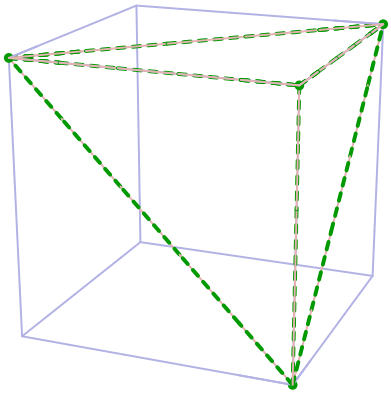
Case 1



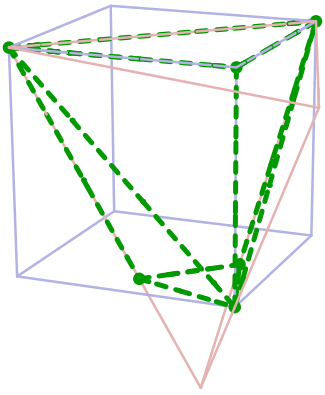
Case 2



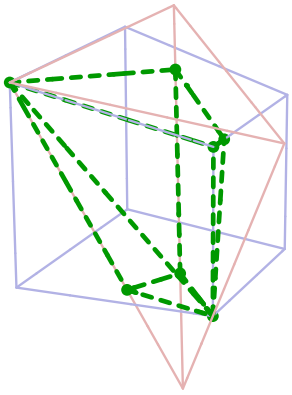
Case 3



Case 4

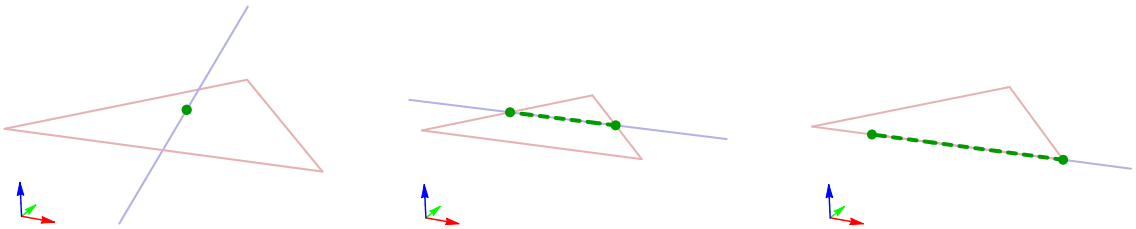


Case 5



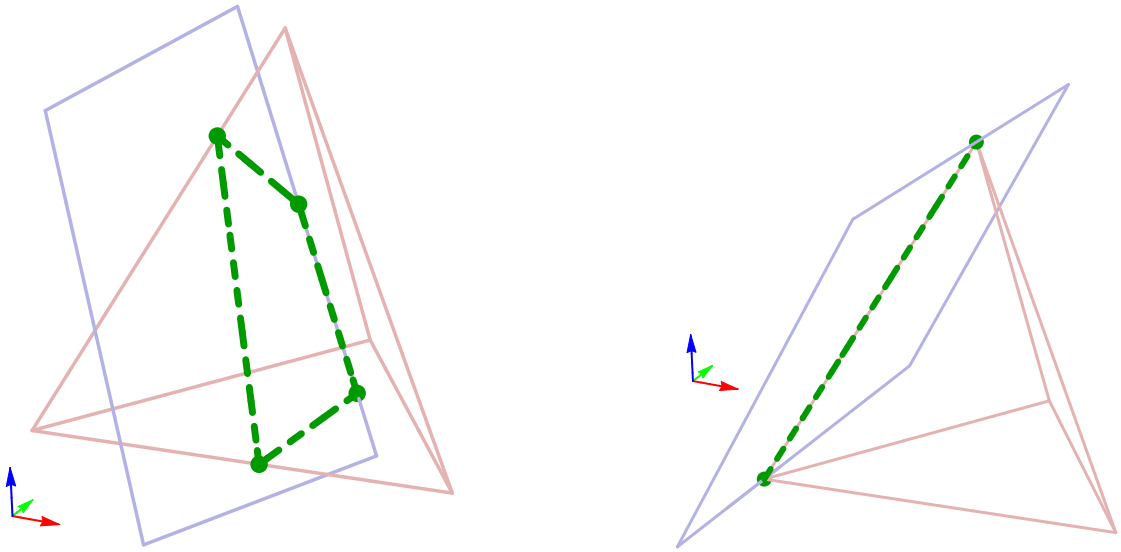
Case 6

Figure A.11: 3D Hexahedron-Tetrahedron intersection



Case 1 Case 2 Case 3

Figure A.12: 3D Line-Triangle intersection



Case 1 Case 2

Figure A.13: 3D Quadrilateral-Tetrahedron intersection

REFERENCES

[1] M. Loève. Van Nostrand, 1955.

[2] N. Aubry, "On the Hidden Beauty of the Proper Orthogonal Decomposition," *Theoret. Comput. Fluid Dynamics*, vol. 2, no. 900265, pp. 339–352, 1991.

[3] M. D. Graham and I. G. Kevrekids, "Alternate Approaches To The Karhunen-Loève Decomposition For Model Reduction And Data Analysis," *Computer Chem. Engng*, vol. 20, no. 5, pp. 495–506, 1996.

[4] M. Rathinam and L. R. Petzold, "A New Look at Proper Orthogonal Decomposition," *SIAM Journal on Numerical Analysis*, vol. 41, no. 5, pp. 1893–1925, 2003.

[5] R. Everson and L. Sirovich, "KarhunenLoeve procedure for gappy data," *JOSA A*, vol. 12, no. 8, pp. 1657–1664, 1995.

[6] D. J. Lucia, P. S. Beran, and W. A. Silva, "Decomposition and Volterra Theory," *Journal of Aricraft*, vol. 42, no. 2, 2005.

[7] D. J. Lucia and P. S. Beran, "Projection methods for reduced order models of compressible flows," *Journal of Computational Physics*, vol. 188, no. 1, pp. 252–280, June 2003.

[8] J. Brigham and W. Aquino, "Inverse viscoelastic material characterization using POD reduced-order modeling in acousticstructure interaction," *Computer Methods in Applied Mechanics and ...*, vol. 198, no. 9-12, pp. 893–903, Feb. 2009.

[9] B. Raghavan, L. Xia, P. Breitkopf, A. Rassineux, and P. Villon, "Towards simultaneous reduction of both input and output spaces for interactive simulation-

based structural design," *Computer Methods in Applied Mechanics and Engineering*, vol. 265, pp. 174–185, Oct. 2013.

[10] I. Kalashnikova, "A stable Galerkin reduced order model for coupled fluidstructure interaction problems," *International Journal for ...*, vol. 95, no. 2, pp. 121–144, 2013.

[11] M. F. Barone, I. Kalashnikova, D. J. Segalman, and H. K. Thornquist, "Stable Galerkin reduced order models for linearized compressible flow," *Journal of Computational Physics*, vol. 228, no. 6, pp. 1932–1946, Apr. 2009.

[12] B. Freno and P. Cizmas, "A proper orthogonal decomposition method for nonlinear flows with deforming meshes," in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, no. January, 2013, pp. 1–21.

[13] A. Paul-Dubois-Taine and D. Amsallem, "An adaptive and efficient greedy procedure for the optimal training of parametric reduced-order models," *International Journal for Numerical Methods in Engineering*, vol. 94305, pp. 1–32, 2014.

[14] M. Xiao and P. Breitkopf, "Constrained Proper Orthogonal Decomposition based on QR-factorization for aerodynamical shape optimization," *Applied mathematics ...*, vol. 223, pp. 254–263, Oct. 2013.

[15] E. Iuliano and D. Quagliarella, "Proper Orthogonal Decomposition, surrogate modelling and evolutionary optimization in aerodynamic design," *Computers & Fluids*, vol. 84, pp. 327–350, Sept. 2013.

[16] D. Amsallem, M. Zahr, Y. Choi, and C. Farhat, "Design Optimization Using Hyper-Reduced-Order Models," *submitted for publication*, pp. 1–16, 2013.

[17] T. Bui-Thanh, "Model-constrained optimization methods for reduction of parameterized large-scale systems," Ph.D. dissertation, MIT, 2007.

[18] D. Binion and X. Chen, "A Krylov enhanced proper orthogonal decomposition method for efficient nonlinear model reduction," *Finite Elements in Analysis and Design*, vol. 47, no. 7, pp. 728–738, July 2011.

[19] Z. Wang, I. Akhtar, J. Borggaard, and T. Iliescu, "Two-level discretizations of nonlinear closure models for proper orthogonal decomposition," *Journal of Computational Physics*, vol. 230, no. 1, pp. 126–146, Jan. 2011.

[20] K. Washabaugh and D. Amsallem, "Nonlinear model reduction for CFD problems using local reduced-order bases," *42nd AIAA Fluid Dynamics ...*, pp. 1–16, 2012.

[21] D. Amsallem, M. Zahr, and C. Farhat, "Nonlinear model order reduction based on local reducedorder bases," *International Journal for ...*, 2012.

[22] I. Kalashnikova and M. Barone, "Efficient nonlinear proper orthogonal decomposition/Galerkin reduced order models with stable penalty enforcement of boundary conditions," *International Journal for Numerical ...*, no. May, pp. 1337–1362, 2012.

[23] M. Vitse, D. Néron, and P. Boucard, "A Time-Space-Parameters Proper Generalised Decomposition Approach for Nonlinear Computational Mechanics," in *International conference on computational structures Technology*, 2014, pp. 1–20.

[24] D. Amsallem and C. Farhat, "Interpolation Method for Adapting Reduced-Order Models and Application to Aeroelasticity," *AIAA journal*, vol. 46, no. 7, pp. 1–29, July 2008.

[25] A. Edelman, T. Arias, and S. Smith, "The geometry of algorithms with orthogonality constraints," *SIAM journal on Matrix Analysis and ...*, vol. 20, no. 2, pp. 303–353, 1998.

[26] P. Absil, R. Mahony, and R. Sepulchre, "Riemannian geometry of Grassmann manifolds with a view on algorithmic computation," *Acta Applicandae Mathematica*, vol. 80, no. 2, pp. 199–220, 2004.

[27] "COMSOL."

[28] "MATLAB."

[29] H. B. Nielson, S. N. Lophaven, and J. Sondergaard, "DACE A Matlab Kriging Toolbox," 2002. [Online]. Available: http://www.imm.dtu.dk/~hbni/dace/

[30] P.-A. Absil, R. Mahony, and R. Sepulchre. Princeton University Press, 2008.

[31] W. R. Inc., "Mathematica, Version 11.1," champaign, IL, 2017.

[32] X.-X. Cai, B. Jiang, and G. Liao, "Adaptive grid generation based onthe least-squares finite-element method," *Computers & Mathematics with Applications*, vol. 48, no. 7, pp. 1077 – 1085, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0898122104003463