Galerkin Projections Between Finite Element Spaces

Ross A. Thompson

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Jeffrey T. Borggaard, Chair
Slimane Adjerid
Matthias Chung

May 6, 2015
Blacksburg, Virginia

Keywords: Finite element methods, adaptive mesh refinement, multi-mesh interpolation

Galerkin Projections Between Finite Element Spaces

Ross A. Thompson

(ABSTRACT)

Adaptive mesh refinement schemes are used to find accurate low-dimensional approximating spaces when solving elliptic PDEs with Galerkin finite element methods. For nonlinear PDEs, solving the nonlinear problem with Newton's method requires an initial guess of the solution on a refined space, which can be found by interpolating the solution from a previous refinement. Improving the accuracy of the representation of the converged solution computed on a coarse mesh for use as an initial guess on the refined mesh may reduce the number of Newton iterations required for convergence. In this thesis, we present an algorithm to compute an orthogonal L^2 projection between two dimensional finite element spaces constructed from a triangulation of the domain. Furthermore, we present numerical studies that investigate the efficiency of using this algorithm to solve various nonlinear elliptic boundary value problems.

Contents

1 Introduction **1**

2 Background **3**

2.1 Finite Element Methods 3

2.1.1 Weak Formulation of the PDE Problem 3

2.1.2 Finite Element Approximation 5

2.1.3 Constructing Piecewise Polynomial Bases 8

2.2 Adaptive Mesh Refinement 10

2.3 Newton's Method for Nonlinear PDEs 12

3 Orthogonal Projection Between Meshes **14**

3.1 Formulation of the Projection Problem 14

3.2 Computing the Projection Matrix 15

3.3 Implementation Details 16

4 Numerical Studies **21**

4.1 Bratu's Equation 21

4.2 The Bratu-Gelfand Problem 25

4.3 Solutions Near Singular Points 26

5 Conclusion **29**

List of Figures

2.1 Lagrange elements of order 1, 2, and 3 and a Pascal triangle representation of the polynomial space covered by each element. 9

2.2 Triangulations of $[0,1]^2$ constructed using an adaptive mesh refinement scheme while computing solutions to Bratu's Equation (4.1). 11

3.1 Meshes whose element intersections are dominated by edge intersections (left) and element containment (right) 17

3.2 Search regions defined by circumscribed circles of a right and an obtuse triangle. 18

4.1 Total Newton iterations required to compute a solution to (4.1) with $f = (x,y) = -2c\left(x(x-1)+y(y-1)\right) - e^{cx(x-1)y(y-1)}$ with c varying from 50 to 100. Zero iterations indicates the method failed to converge. 25

List of Tables

4.1 Interpolation vs projection errors on meshes of size $h = 2^{-k}$. 23

4.2 Interpolation vs projection errors after one refinement. 23

4.3 Computation time (in seconds) to interpolate from a mesh of size $h = 2^{-k}$ to a mesh refinement that is of size $2^{-(k+1)}$. 23

4.4 Iterations required for convergence of Newton's method at each refinement. . 24

4.5 Total computation time (in seconds) required for each refinement. 24

4.6 Total Newton iterations required to compute a solution to (4.3) with λ ranging from 6.8 to 6.808. 26

4.7 Iterations required for Newton's method to converge when $\lambda = 10$ and k varies from 1.15 to 1. 27

4.8 Time (in secs) required to compute the finite element solution on each refinement when $\lambda = 10$ and k varies from 1.15 to 1. 28

4.9 Time (in secs) required to compute the interpolation and the projection on each refinement when $\lambda = 10$ and $k = 1$. 28

4.10 Time (in secs) required for Newton's method to converge on each refinement when $\lambda = 10$ and $k = 1$. 28

Chapter 1

Introduction

Finite element methods are frequently used to approximate solutions to partial differential equations using finite-dimensional spaces. These spaces are formed by discretizing the domain of a problem by partitioning it into a set of elements, usually convex polygons, and then defining a set a basis functions, usually piecewise polynomials, over these elements.

By representing the approximate solution with a finite-dimensional basis, we are able to express the solution to a linear partial differential equation as the solution to a linear system of equations. However, the error in the approximation is dependent upon the size of the elements used to define a finite element space, and increasing the number of elements in a partition of the domain rapidly increases the dimension of the approximating space, particularly on higher dimensional domains. Thus, reducing the error in the finite element solution rapidly increases the computational costs necessary to find the solution.

One technique used to limit the dimension of the finite element space while reducing the error in the solution is adaptive mesh refinement. This technique attempts to construct a finite element space that is well-suited to approximate the solution to a particular problem through an iterative process. An initial low-dimensional space is defined over a coarse mesh, and an approximate solution is computed on this space. Then, an *a posteriori* error estimate is used to find regions of the domain where the error in the approximation is high. Additional elements are added to these regions, forming a higher-dimensional approximation space, and the process is repeated.

Adaptive mesh refinement is particularly effective when solving nonlinear partial differential equations. Using Newton's method, the solution to a nonlinear partial differential equation can be found by computing solutions to a sequence of linear partial differential equations. Each Newton iteration requires solving a system of equations whose size is equal to the dimension of the approximating space. Furthermore, Newton's method requires an initial guess for the solution, and the speed of convergence for the method is dependent on the accuracy of this initial guess. Using adaptive mesh refinement, an initial guess can be found

by interpolating the solution from a previous finite element space onto the new refinement. In most cases, improving the accuracy of this interpolant should provide a more accurate initial guess to Newton's method and potentially reduce the number of iterations needed for the method to converge.

This thesis presents an algorithm for computing the orthogonal L^2 projection between two finite element spaces defined on the same domain. We discuss the computational difficulties associated with finding this projection and provide a detailed implementation of our algorithm. Finally, we investigate the computational efficiency of using this projection as the inter-mesh interpolant in an adaptive mesh refinement scheme through several numerical experiments.

Chapter 2

Background

2.1 Finite Element Methods

Finite element methods are a common approach for finding finite dimensional approximations to partial differential equations. Because finite element methods are based on finding solutions in a space of piecewise polynomial functions, they provide easy implementation for irregular domains and allow for flexible adaptation of these functions by rearranging the simplices used to construct them. As we will see, the advantages in obtaining a better basis for representing the solution of a PDE come with a challenge to accurately represent functions in bases constructed using different sets of simplices. For simplicity, we restrict our attention to polyhedral domains.

2.1.1 Weak Formulation of the PDE Problem

We consider the approximation to Poisson's equation using finite element methods. Give a domain $\Omega \in \mathbb{R}^2$ and a source term $f : \mathbb{R}^2 \to \mathbb{R}$, we seek the solution $u : \mathbb{R}^2 \to \mathbb{R}$ to

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y) \qquad \text{in } \Omega = (0, 1)^2 \tag{2.1}$$

$$u \equiv 0 \qquad \text{on } \partial\Omega. \tag{2.2}$$

If the solution u is approximated using a piecewise polynomial function, we have no guarantee that this approximation will be twice differentiable (unless we use Hermite basis functions). Yet we wish to claim that such an approximation would "solve" this partial differential equation over the finite element space. To make sense of this claim, we reformulate the

partial differential equation using integration by parts. If u is a solution to (2.1) and v is any sufficiently smooth function such that $v \equiv 0$ on $\partial\Omega$ then

$$-\int_\Omega \frac{\partial^2 u}{\partial x^2}(x,y)v(x,y)dxdy - \int_\Omega \frac{\partial^2 u}{\partial y^2}(x,y)v(x,y)dxdy = \int_\Omega f(x,y)v(x,y)dxdy. \tag{2.3}$$

Integration by parts and application of the homogenous Dirichlet conditions yields the equation

$$\int_\Omega \frac{\partial u}{\partial x}(x,y)\frac{\partial v}{\partial x}(x,y)dxdy + \int_\Omega \frac{\partial u}{\partial y}(x,y)\frac{\partial v}{\partial y}(x,y)dxdy = \int_\Omega f(x,y)v(x,y)dxdy. \tag{2.4}$$

Therefore, we now seek the function u belonging to the Hilbert space $H_0^1(\Omega)$, the space of functions that vanish on the boundary and have one weak derivative, that solves

$$\int_\Omega \nabla u(x,y)\cdot\nabla v(x,y)dxdy = \int_\Omega f(x,y)v(x,y)dxdy \tag{2.5}$$

for every $v \in H_0^1(\Omega)$, which we denote by the weak formulation of (2.1). Note the weak formulation only requires u and v to have first partial derivatives, and it reformulates the partial differential equation in terms of an $L^2(\Omega)$ inner product.

By the Cauchy-Schwarz inequality, we can bound these inner products by

$$\int_\Omega \frac{\partial u}{\partial x}(x,y)\frac{\partial v}{\partial x}(x,y)dxdy \le \left\|\frac{\partial u}{\partial x}\right\|_{L^2(\Omega)}\left\|\frac{\partial v}{\partial x}\right\|_{L^2(\Omega)} \tag{2.6}$$

$$\int_\Omega \frac{\partial u}{\partial y}(x,y)\frac{\partial v}{\partial y}(x,y)dxdy \le \left\|\frac{\partial u}{\partial y}\right\|_{L^2(\Omega)}\left\|\frac{\partial v}{\partial y}\right\|_{L^2(\Omega)} \tag{2.7}$$

$$\int_\Omega f(x,y)v(x,y)dxdy \le \|f\|_{L^2(\Omega)}\|v\|_{L^2(\Omega)}. \tag{2.8}$$

Thus to ensure the terms in (2.5) are well defined and finite, we see that the spaces $u, v \in H_0^1(\Omega)$ and $f \in L^2(\Omega)$ are natural choices for the weak form of the problem [1].

[1] In fact we can require even less regularity on f, as the only requirement is that the right hand side of (2.5) make sense. Functions $f \in H^{-1}(\Omega)$, the dual space of $H_0^1(\Omega)$, would suffice. In this case, the right hand side of (2.8) becomes $\|f\|_{H^{-1}(\Omega)}\|v\|_{H^1(\Omega)}$.

This example can be extended to the general case of two-dimensional second order linear elliptic partial differential equations. Following the discussion in Larson et al. [10], consider a PDE of the form

$$-\nabla \cdot (a\nabla u) + b \cdot \nabla u + cu = f \qquad \text{in } \Omega \subset \mathbb{R}^2 \tag{2.9}$$

$$u = 0 \qquad \text{on } \partial\Omega, \tag{2.10}$$

where Ω is an open domain with Lipschitz boundary, $u : \Omega \to \mathbb{R}$, $u \in H_0^1(\Omega)$, and $a_{ij}, b_i, c, f \in L^2(\Omega)$. Note that for an elliptic partial differential equation the matrix a must be symmetric positive definite. Let $v \in H_0^1(\Omega)$ be any arbitrary test function. Multiplying (2.9) by v and integrating by parts, we construct the weak formulation of the problem, find $u \in H_0^1(\Omega)$ such that

$$\langle a\nabla u, \nabla v\rangle + \langle b \cdot \nabla u, v\rangle + \langle cu, v\rangle = \langle f, v\rangle \tag{2.11}$$

holds for every $v \in H_0^1(\Omega)$. For abbreviation, we express the bilinear and linear forms in (2.11) as

$$\begin{aligned} a(u, v) &= \langle a\nabla u, \nabla v\rangle + \langle b \cdot \nabla u, v\rangle + \langle cu, v\rangle \\ b(v) &= \langle f, v\rangle. \end{aligned}$$

We say that u is a weak solution to (2.9) if it satisfies (2.11).

2.1.2 Finite Element Approximation

To compute a solution, u to the weak formulation developed in the previous section, we must be able to construct an approximation u^h from a finite-dimensional function space. For demonstration purposes, we again consider the two-dimensional Poisson problem defined in (2.1). To satisfy the boundary condition $u \equiv 0$ on $\partial\Omega$, we require $u \in V = H_0^1(\Omega)$. Let $V^h \subset V$ be a finite-dimensional subspace with basis $\{\phi_1, \phi_2, ..., \phi_n\}$. Then we can define an approximation $u^h \in V^h$ by

$$u^h(\mathbf{x}) = \sum_{i=1}^{n} u_i \phi_i(\mathbf{x}), \tag{2.12}$$

where $\mathbf{u} = (u_1, u_2, ..., u_n)$ is a real valued coefficient vector. Replacing u with u^h in the weak formulation (2.5) yields

$$\sum_{i=1}^{n} u_i \int_\Omega \nabla\phi_i(x,y) \cdot \nabla v(x,y) dxdy = \int_\Omega f(x,y)v(x,y)dxdy. \tag{2.13}$$

If we similarly choose $v \in V^h$, then the condition that (2.12) is satisfied for each $v \in V$ is equivalent to the condition that

$$\sum_{i=1}^{n} u_j \int_\Omega \nabla\phi_j(x,y) \cdot \nabla\phi_i(x,y) dxdy = \int_\Omega f(x,y)\phi_i(x,y)dxdy \quad \text{for } \ i = 1, 2, ..., n. \tag{2.14}$$

Thus to find a function u^h that satisfies (2.13), we must only solve a system of n equations with n unknowns. Letting $K \in \mathbb{R}_{n\times n}$ be given by $K_{ij} = \int_\Omega \nabla\phi_j(x,y) \cdot \nabla\phi_i(x,y)dxdy$ and $b_i = \int_\Omega f(x,y)\phi_i(x,y)dxdy$, we have the matrix formulation

$$K\mathbf{u} = \mathbf{b}. \tag{2.15}$$

In the general case (2.11), we have the condition

$$a(u^h, v) = b(v) \qquad \forall v \in V^h, \tag{2.16}$$

which is equivalent to the requirement

$$a(u^h, \phi_i) = b(\phi_i) \qquad \text{for} \quad i = 1, 2, ..., n. \tag{2.17}$$

So we can similarly define $K \in \mathbb{R}_{n\times n}$ by $K_{ij} = a(\phi_j, \phi_i)$ and $\mathbf{b}_i = b(\phi_i)$ to again express u^h as a solution to (2.15). Thus, u^h may be found by computing the solution to a linear system of equations.

We now have a finite-dimensional approximation to the weak solution, but we would like to know how good the approximation u^h is relative to the space V^h. To do this, we must first establish the following fact regarding approximations in Hilbert spaces [11].

Theorem 2.1.1. *Let $\mathcal{A}$ be a linear subspace of a Hilbert space $\mathcal{H}$, and let f be any element of $\mathcal{H}$. The point $p^* \in \mathcal{A}$ is the best approximation from $\mathcal{A}$ to f if and only if the error $e^* = f - p^*$ satisfies the orthogonality conditions*

$$\langle e^*, p \rangle = 0 \qquad \forall p \in \mathcal{A}. \tag{2.18}$$

We call the point p^* the *orthogonal projection* of f onto $\mathcal{A}$. Letting $e = u - u^h$ and subtracting the finite element equations (2.16) from the weak form (2.11) yields

$$a(e, v) = 0 \qquad \forall v \in V^h. \tag{2.19}$$

Therefore the error is orthogonal to the approximation space V^h with respect to an inner product generated by $a(\cdot, \cdot)$. If we let V^h be a finite element space, as we will describe in Section 2.1.3, this orthogonality condition gives rise to an *a priori* error estimate for finite element approximations [10].

Theorem 2.1.2. *(Cea's Lemma) The error e satisfies the best approximation result*

$$\|e\|_V \leq \frac{C_a}{m} \min_{v \in V^h} \|u - v\|_V \tag{2.20}$$

for some constants C_a, m. The latter incorporates the coercivity of $a(\cdot, \cdot)$ on V.

Letting v be the Clement interpolation [10] of u, and assuming extra smoothness of the solution $u \in H^2(\Omega)$, we have the estimate $\|u - v\|_V \leq Ch|u|_{H^2(\Omega)}$. In this case, we have the following estimate.

Theorem 2.1.3. *The error e satisfies the* a priori *estimate*

$$\|e\|_V \leq Ch|u|_{H^2(\Omega)} \tag{2.21}$$

where h is the mesh size of V^h and C is independent of h.

Thus the error in the finite element solution is at worst directly proportional to the mesh size h, and the finite element approximation u^h will theoretically converge to the weak solution u. As shown below, the choice of V^h can lead to a better estimate of the approximation error by improving the estimate of the best approximation term in (2.20). The finite element method builds approximating subspaces in two steps. The first is to subdivide the domain into subdomains and the second is to construct polynomial functions over each subdomain. We present this approach below.

2.1.3 Constructing Piecewise Polynomial Bases

So far, we have only presented V^h as a subspace of our solution space V. We now wish to construct a space of finite-dimensional piecewise polynomial functions to serve as our approximation space. Following the discussion in Larson et al. [10], we choose a simplex, T, and construct a partition $\mathcal{T} = \{T\}$ of the domain $\Omega \subset \mathbb{R}^d$ into simplices of type T. In this thesis, we restrict our discussion to triangulations of two-dimensional polyhedral domains. We next define a polynomial function space P on $\mathcal{T}$ with a basis $\{\phi_j\}_{j=1}^n$. Finally, we establish a set of linear functionals $\{\ell_i\}_{i=1}^n$ on P such that $\ell_i(\phi_j) = \delta_{i,j}$ for $i, j = 1, 2, ..., n$.

A common choice of functionals defines

$$\ell_i(v) = v(N_i) \qquad i = 1, 2, ..., n,$$

where $v \in P$ and N_i are Lagrange nodal points. This gives rise to the family of Lagrange interpolating polynomials over T and ensures continuity of functions in our approximation space V^h.

The shape functions for the linear Lagrange finite element can be derived by considering shape functions over a reference element $\hat{T}$ with vertices $(0,0), (0,1)$, and $(1,0)$. Using the standard basis $\{1, x_1, x_2\}$ for two-dimensional linear polynomials, we can express each shape function as a linear combination

$$\phi_i = c_{i,0} + c_{i,1}x_1 + c_{i,2}x_2.$$

We can solve for the coefficients using the requirement $\ell_i(\phi_j) = \delta_{i,j}$. For the first shape function, we can compute

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \ell_1(\phi_1) \\ \ell_2(\phi_1) \\ \ell_3(\phi_1) \end{pmatrix} = \begin{pmatrix} \ell_1(1) & \ell_1(x_1) & \ell_1(x_2) \\ \ell_2(1) & \ell_2(x_1) & \ell_2(x_2) \\ \ell_3(1) & \ell_3(x_1) & \ell_3(x_2) \end{pmatrix} \begin{pmatrix} c_{1,0} \\ c_{1,1} \\ c_{1,2} \end{pmatrix}. \tag{2.22}$$

The nodes for the reference element are given by its vertices. So the requirement $L_i(v) = v(N_i)$ allows us to compute $L_1(1) = 1$, $L_1(x_1) = x_1(0,0) = 0$, and $L_1(x_2) = x_2(0,0) = 0$. Similarly computing the values of L_2 and L_3 gives us the linear equation

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_{1,0} \\ c_{1,1} \\ c_{1,2} \end{pmatrix}, \tag{2.23}$$

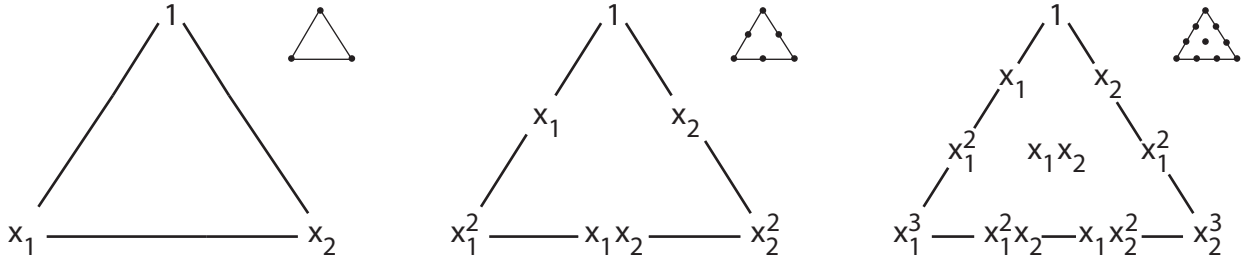

Figure 2.1: Lagrange elements of order 1, 2, and 3 and a Pascal triangle representation of the polynomial space covered by each element.

which gives the coefficients $c_{1,0} = 1$, $c_{1,1} = -1$, and $c_{1,2} = -1$. Therefore $\phi_1(\mathbf{x}) = 1 - x_1 - x_2$. Using e_2 and e_3, we can similarly compute $\phi_2(\mathbf{x}) = x_1$ and $\phi_3(\mathbf{x}) = x_2$. Note that the shape functions for arbitrary triangular elements may be found by an affine mapping from the reference element.

For higher order polynomial spaces, we introduce additional nodes by partitioning the edges of $\hat{T}$ into p segments of equal length where p is the order of the polynomial space. Interior nodes are added by extending the pattern of nodes along the edges in order to form an equally spaced lattice of nodal points with $n = \frac{(p+1)(p+2)}{2}$ total nodes as demonstrated in Figure 2.1. Note that the number of nodes in each reference element correspond to the number of basis functions required to form the space of p-degree polynomials. Each node N_i corresponds to a shape function ϕ_i, which can be determined in the same process we applied to the first-order case.

Furthermore, we will restrict the triangulations of Ω to a set of admissible partitions with the following properties [3]

1. $\bar{\Omega} = \cup_{i=1}^{N} T_i$.
2. If $T_i \cap T_j$ consists of exactly one point, then it is a common vertex of T_i and T_j.
3. If for $i \neq j$, $T_i \cap T_j$ consist of more than one point, then $T_i \cap T_j$ is a common edge of T_i and T_j.

Finally, using Lagrange basis functions on admissible triangulations, we can establish a tighter error estimate than we presented in Theorem 2.1.3 [3].

Theorem 2.1.4. *Let V^h be approximating spaces consisting of piecewise polynomials of degree $t-1$ on a family of shape-regular admissible triangulations, $\mathcal{T}_h$, of Ω. Let $t \geq 2$ and let κ be the shape parameter associated with $\mathcal{T}_h$. Then there exists a constant c dependent on Ω, κ, and t (but not h or u), such that*

$$\|u - I_h u\|_{H^m(\Omega)} \leq c h^{t-m} |u|_{H^t(\Omega)} \qquad 0 \leq m \leq t, \tag{2.24}$$

where $u \in H^t(\Omega)$ and I_h denotes interpolation by a piecewise polynomial of degree $t-1$.

Using this interpolation bound in Cea's Lemma yields the following estimate.

Theorem 2.1.5. *Let $u \in H_0^t(\Omega)$ be the weak solution to (2.9) and let $u^h \in H_0^t(\Omega)$ be the finite element solution to (2.16). Let V^h be defined as in Theorem 2.1.4. Then the error $e = u - u^h$ satisfies the estimate*

$$\|e\|_{H^m(\Omega)} \leq C h^{t-m} |u|_{H^t(\Omega)} \qquad 0 \leq m \leq t, \tag{2.25}$$

where the constant C is dependent on u, $\mathcal{T}_h$, t, and Ω.

2.2 Adaptive Mesh Refinement

From our error estimate in Theorem 2.1.5, for a fixed polynomial order, the error in our finite element approximation is dependent on the mesh size and a constant that depends on the structure of the mesh. To reduce the error in our solution, we could simply reduce the size of our mesh, but doing so rapidly increases the dimension our approximation space V^h, especially in higher spatial dimensions. Since computing the approximation requires solving an $n \times n$ system of equations where $n = dim(V^h)$, decreasing h can significantly increase the computation time required to find a solution.

To limit the dimension of V^h while still obtaining good approximations of the solution, we can use *a posteriori* error estimates on a very coarse mesh to determine those parts of the domain that require more nodal points to accurately approximate the solution. By adding more elements of smaller diameter where the error is high and increasing the size of the elements where error is low, we can reduce the error in the finite element approximation while limiting the dimension of V^h. Since the error estimates also converge with refinement, we repeat this process over several refinements, and ultimately construct a mesh that is specifically tailored to well approximate the solution u.

The effectiveness of an adaptive mesh refinement scheme is dependent on the accuracy of the *a posteriori* error estimate used to construct the refinements. We use a feature-based

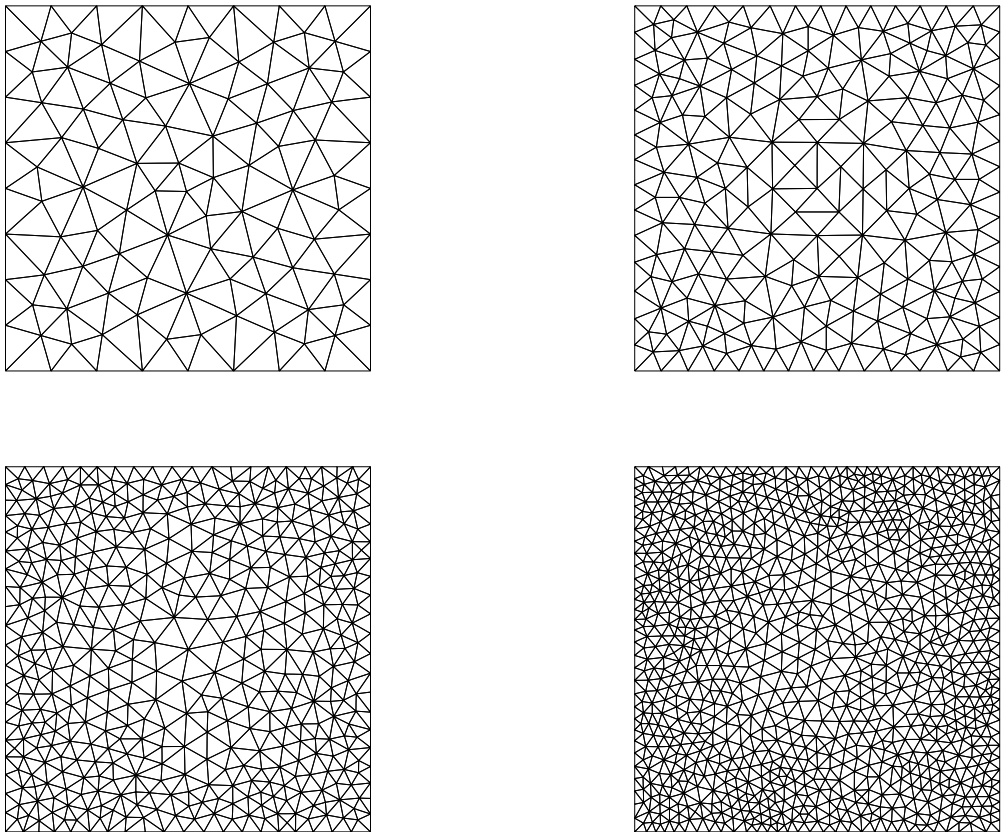

Figure 2.2: Triangulations of $[0,1]^2$ *constructed using an adaptive mesh refinement scheme while computing solutions to Bratu's Equation (4.1).*

estimate devised by Zienkiewicz and Zhu [13] that uses an estimate for the error of the gradient to approximate the relative error of the finite element solution. This error estimate compares the difference between the finite element solution gradient, which is discontinuous along element edges, and the least-squares projection of the gradient onto the continuous finite element space V^h. This estimate of the H^1-seminorm has the effect of identifying regions of the solution that have high curvature.

2.3 Newton's Method for Nonlinear PDEs

We consider the role of finite element methods in computing approximate solutions to nonlinear partial differential equations. Let V and U be two Hilbert spaces and let $\mathcal{N} : U \to V$ be a nonlinear differential operator that is Fréchet differentiable. We wish to find $u \in U$ such that

$$\mathcal{N}(u) = f. \tag{2.26}$$

The Fréchet derivative of $\mathcal{N}(\cdot)$ at $u \in U$ is defined as the linear operator $\mathcal{N}'(u)$ from U to V. Newton's method applied to (2.26) begins with an initial guess $u_0 \in U$ and produces the sequence

$$u_{n+1} = u_n - \left[\mathcal{N}'(u_n)\right]^{-1} \left(\mathcal{N}(u_n) - f\right).$$

This is implemented as first solving

$$\left[\mathcal{N}'(u_n)\right] s_n = f - \mathcal{N}(u_n) \tag{2.27}$$

for the Newton step s_n and then setting $u_{n+1} = u_n + s_n$. Local convergence is guaranteed if (2.27) is uniquely solvable at the solution u of (2.26), the Fréchet derivative is locally Lipschitz continuous at the solution u, and

$$\left\|\mathcal{N}'(\bar{u}) - \mathcal{N}'(\bar{\bar{u}})\right\| \leq L\|\bar{u} - \bar{\bar{u}}\|$$

for all $\bar{u}, \bar{\bar{u}}$ in a neighborhood of u, where $L > 0$ is the Lipschitz constant. If the initial guess is sufficiently close, $\|u - u_0\| \leq \delta$ for some $\delta > 0$, Newton's method has the q-quadratic rate of convergence

$$\|u - u_{n+1}\| \leq C \|u - u_n\|^2.$$

To apply Newton's method, we need to approximate the solution to (2.27) and apply the finite element method for this step. Consider a sequence of refined meshes and apply Newton's method (in $\mathbb{R}^n$) on each mesh until a given convergence tolerance is met. For an important class of nonlinear PDEs we have the result that these approximate solutions converge to an approximation to the solution of (2.26) that meets the same convergence tolerance and so Newton's method ultimately behaves in a mesh-independent way. In other words, the number of terms in the Newton sequence required to meet this tolerance converges with mesh refinement (see [1, 2]).

Note that the convergence of Newton's method requires an initial guess that is sufficiently close to the true solution to ensure a q-quadratic rate of convergence.

> So by choosing u_0 carefully, we can potentially reduce the number of Newton iterations required to solve a nonlinear problem.

When solving a nonlinear problem using an adaptive mesh refinement scheme, we can use information about the solution on the previous mesh to find a good initial guess for the solution on the refined mesh. Typically, direct interpolation between the two meshes is used

$$u^{h+1}(x_i) = u^h(x_i) \qquad i = 1, 2, ..., n(h+1), \tag{2.28}$$

where x_i are the nodes of the refined mesh (indicated by the $(h+1)$). However, this interpolant may not be accurate enough to guarantee that Newton's method converges on the refined mesh, or that Newton's method will converge q-quadratically. One of the motivations for the algorithm developed in this thesis is to find the best representation of the solution u^h on the mesh defined for V^h in the refined finite element space V^{h+1}. As we have stated in Theorem 2.1.1, this is defined by the orthogonal projection. Our algorithm is presented in the next chapter.

Chapter 3

Orthogonal Projection Between Meshes

3.1 Formulation of the Projection Problem

We wish to find an interpolant between finite element spaces that will be optimal in the L^2-norm. While we intend to use this interpolant with an adaptive mesh refinement scheme, we set up this problem in the general case of interpolation between two arbitrary finite element spaces defined on the same polyhedral domain Ω.

Let $V^\phi(\Omega)$ and $V^\psi(\Omega)$ be finite-dimensional subspaces of $L^2(\Omega)$ for some $\Omega \subset \mathbb{R}^2$ with basis functions $\{\phi_1(x,y), \phi_2(x,y), ..., \phi_n(x,y)\}$ and $\{\psi_1(x,y), \psi_2(x,y), ..., \psi_m(x,y)\}$, respectively. Let $u(x,y) = \sum_{j=1}^{n} u_j \phi_j(x,y)$ be a member of $V^\phi(\Omega)$. From Theorem 2.1.1, we know that the optimal interpolant in the L^2-norm is given by the orthogonal projection, u^p, of u onto $V^\psi(\Omega)$. Then $u^p(x) = \sum_{j=1}^{m} u_j^p \psi_j(x)$ must satisfy the orthogonality condition

$$\langle u - u^p, v \rangle = 0 \qquad \text{for all} \quad v \in V^\psi. \tag{3.1}$$

This is equivalent to the requirement that the error function $u - u^p$ be orthogonal to each basis function in the space $V^\psi(\Omega)$

$$\langle u - u^p, \psi_i \rangle = 0 \qquad \text{for all} \quad i = 1, 2, ..., m. \tag{3.2}$$

Rewriting the error as a linear combination of basis functions and rearranging yields the condition

$$\sum_{j=1}^{m}\langle u_j^p\psi_j,\psi_i\rangle=\sum_{j=1}^{n}\langle u_j\phi_j,\psi_i\rangle \qquad \text{for all} \quad i=1,2,...,m. \tag{3.3}$$

This equation allows us to define the orthogonal projection in terms of a linear system. Given $\mathbf{u}=(u_1,u_2,...,u_n)^T$, basis functions $\{\phi_i\}_{i=1}^n$, and $\{\psi_j\}_{j=1}^m$ we wish to find the linear coefficients $\mathbf{u^p}\in\mathbb{R}^m$ such that

$$M\mathbf{u^p}=P\mathbf{u},$$

where $M_{ij}=\int_\Omega\psi_i(x,y)\psi_j(x,y)dxdy$ for $i,j=1,\dots,m$ and $P_{ij}=\int_\Omega\phi_j(x,y)\psi_i(x,y)dxdy$ for $i=1,\dots m$ and $j=1,\dots n$. The mass matrix M frequently appears in Galerkin finite element methods and can be easily computed since the functions ψ_i and ψ_j are defined on the same set of simplices. However computing the projection matrix P raises several challenges.

3.2 Computing the Projection Matrix

In order to compute the projection matrix, we must first impose some limitations on the function spaces $V^\phi(\Omega)$ and $V^\psi(\Omega)$. We will only consider finite element spaces formed using simplices as elements and Lagrange basis functions. In particular we will focus on meshes formed by Delaunay triangulations. For now, we let $\mathcal{T}^\phi$ and $\mathcal{T}^\psi$ denote the sets of simplices used to define $V^\phi(\Omega)$ and $V^\psi(\Omega)$, respectively.

We consider the computation of the $L^2(\Omega)$ inner product between the basis functions $\phi\in V^\phi(\Omega)$ and $\psi\in V^\psi(\Omega)$. Let $\mathcal{I}_\phi$ be the set of indices where $i\in\mathcal{I}_\phi$ if $T_i\subset\text{supp}(\phi)$, and define $\mathcal{I}_\psi$ similarly. Since each basis function is nonzero only on neighboring elements, the inner product $\int_\Omega\phi(x,y)\psi(x,y)dxdy$ will be nonzero if and only if there exist elements $T_i\in\mathcal{T}^\phi$ and $\bar{T}_j\in\mathcal{T}^\psi$ such that $T_i\cap\bar{T}_j\neq\emptyset$, where i corresponds to an index in $\mathcal{I}_\phi$ and j corresponds to an index in $\mathcal{I}_\psi$.

Therefore if two elements, $T\in\mathcal{T}^\phi$ and $\bar{T}\in\mathcal{T}^\psi$, with vertices corresponding to basis functions $\{\phi_{T_1},\phi_{T_2},\phi_{T_3},...\phi_{T_n}\}\subset V^\phi(\Omega)$ and $\{\psi_{\bar{T}_1},\psi_{\bar{T}_2},\psi_{\bar{T}_3},...,\psi_{\bar{T}_m}\}\subset V^\psi(\Omega)$, do not intersect, then the product $\phi_{T_j}\psi_{\bar{T}_i}$ must be zero on $T\cup\bar{T}$ for $i=1,2,3,...,m$ and $j=1,2,3,...n$. If two elements do intersect, then the product $\phi_{T_j}\psi_{\bar{T}_i}$ must only be nonzero over the region of intersection. Note that the inner product of basis functions ϕ_j and ϕ_i is the sum of the integrals $\int_{T\cap\bar{T}}\phi_{T_j}(x,y)\psi_{\bar{T}_i}(x,y)dxdy$ over all intersections $T\cap\bar{T}$ where ϕ_j corresponds to a vertex of T and ψ_i corresponds to a vertex of $\bar{T}$.

So to compute the projection matrix, we can iterate over the elements $T \in \mathcal{T}^\phi$ and determine which elements $\bar{T} \in \mathcal{T}^\psi$ intersect T. For each region of intersection, we compute $\int_{T\cap\bar{T}} \phi_{T_j}(x,y)\psi_{\bar{T}_i}(x,y)dxdy$ for $i = 1, 2, ..., m$, $j = 1, 2, ...n$ and add this value to $P_{\bar{T}_i T_j}$. Once this process has completed, the computed matrix P must be the desired projection matrix.

Algorithm 1 Projection Matrix Computation

for $T \in \mathcal{T}^\phi$ **do**
  Compute the set O of elements in $\mathcal{T}^\psi$ whose intersection with T is nonempty
  for $\bar{T} \in O$ **do**
    Construct F, the triangulation of $T \cap \bar{T}$
    for $K \in F$ **do**
      for $i = 1, 2, 3$ **do**
        for $j = 1, 2, 3$ **do**
          $P_{ij} = P_{ij} + \int_K \phi_{T_j}(x,y)\psi_{\bar{T}_i}(x,y)dxdy$

3.3 Implementation Details

Our previous algorithm only requires us to be able to efficiently compute the L^2 inner product of Lagrange basis functions defined over a triangular region and to compute sets of elements in $\mathcal{T}^\psi$ that intersect a given element of $\mathcal{T}^\phi$. The former is a common task in finite element methods, but the later requires some additional work.

Before discussing an algorithm to compute element intersections, we must establish some basic facts regarding intersections of convex polygons.

Lemma 3.3.1. *The intersection of any two convex polygons is a convex polygon.*

Lemma 3.3.2. *Let P_1 and P_2 be convex polygons. If an edge of P_1 intersects an edge of P_2 at a single point, then this point of intersection must be a vertex of $P_1 \cap P_2$.*

Lemma 3.3.3. *Let P_1 and P_2 be convex polygons. If a vertex of P_1 is contained in P_2 then it must be a vertex of $P_1 \cap P_2$ and* vice versa.

Theorem 3.3.1. *Let P_1 and P_2 be convex polygons. Let I be the set of points formed by the intersections between edges as defined in Lemma 3.3.2. Let V_1 be the set of vertices of P_1 contained in P_2 and V_2 be the set of vertices of P_2 contained in P_1. Then the vertices of $P_1 \cap P_2$ are given by $I \cup V_1 \cup V_2$.*

From this theorem, in order to determine whether two elements intersect and to compute their region of intersection, we must be able to determine if a point is contained within an

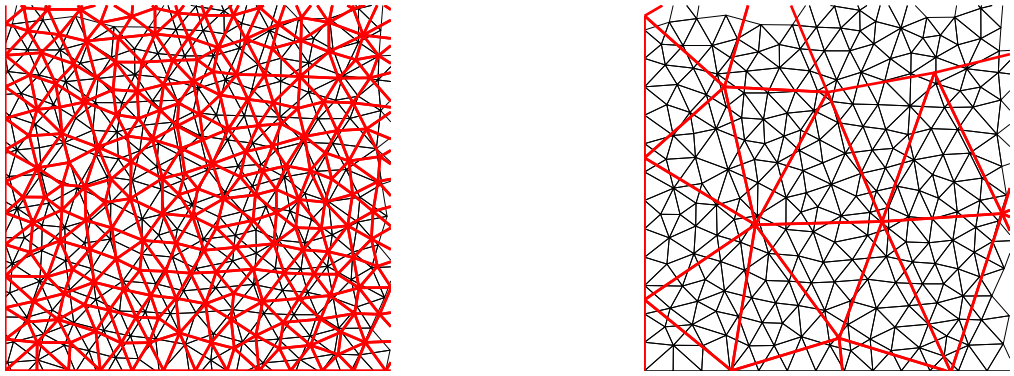

Figure 3.1: Meshes whose element intersections are dominated by edge intersections (left) and element containment (right)

element and we must be able to compute edge intersections between two elements. Point containment can easily be tested using barycentric coordinates. For line segment intersections, we use an algorithm developed by Ronald Goldman [7].

While we now have the tools to compute intersection sets by brute force, we wish to be able to reduce the number of element-to-element intersection tests in our algorithm. One approach would be to compute the set of all edge intersections, for example using the Bentley-Ottmann algorithm, and then attempt to group these intersections according to their corresponding element edges. However, this approach would fail if elements on one mesh could be contained entirely within an element on another mesh, leading to no edge intersections between the elements. We could potentially use a line-intersection strategy as a first pass that is then supplemented with vertex containment checks on neighboring elements, but without knowing the structures of the two meshes, it is impossible to know how effective such a strategy would be.

For example, if $\mathcal{T}^\phi$ and $\mathcal{T}^\psi$ contain a similar number of elements of approximately the same size, as in the left example in Figure 3.1, then the element intersections will be dominated by edges intersections and so a line segment intersection approach would be highly efficient. However, suppose the mesh size of $\mathcal{T}^\phi$ is several times larger than that of $\mathcal{T}^\psi$, as in the right example in Figure 3.1, or that $\mathcal{T}^\psi$ is a refinement of $\mathcal{T}^\phi$ that preserves the parent structure. In either case, we would expect most of the element intersections to be cases where one element is entirely contained within another, and so computing a set of line-intersections would take a large amount of computational time and would reveal comparatively little information about the set of element intersections.

In contrast, we could reduce the number of intersection tests by taking a vertex-centric approach. From the location of a vertex, we could attempt to compute a region in which all

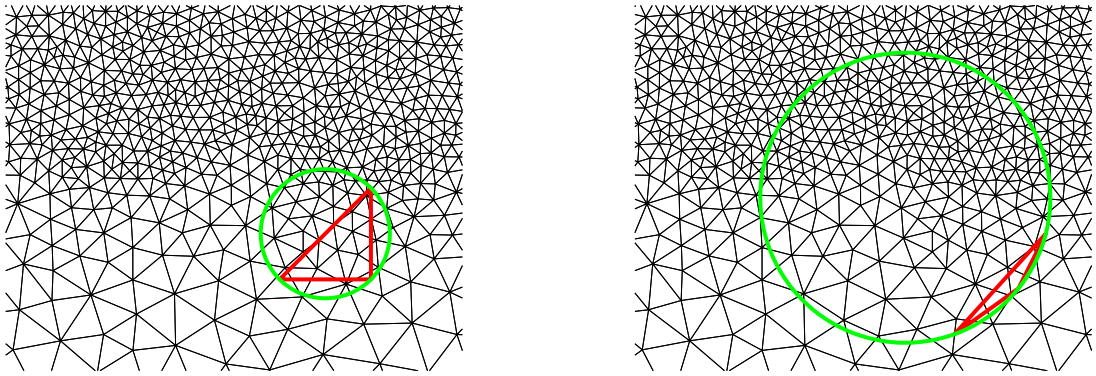

Figure 3.2: Search regions defined by circumscribed circles of a right and an obtuse triangle.

intersecting elements must exist, and then test each element within that region for intersection. This approach would require a secondary data structure, such as a specialized region quadtree or a polygonal map quadtree, that allows us to efficiently retrieve all elements from a mesh lying within a specified region. As with the line segment intersection approach, this would require considerable pre-processing time and additional memory overhead.

Furthermore, we would need to be able to compute a search region from a given vertex or element such that all possible intersecting elements must lie within that search region. To overcome the computational overhead of the pre-processing, we would need this search region to significantly reduce the number of elements tested for intersection compared to the brute force approach. This is possible if we are given regularity requirements on the meshes and we know the ratio of the mesh diameters. However, for two arbitrary meshes, we have no guarantee that a search region would be efficient in reducing element intersection tests.

For example, suppose we define the search region for an element $T \in \mathcal{T}^\phi$ by the interior of the circumscribed circle C_T as in Figure 3.2. If T, shown in red, is close to equilateral, then the search region will include a relatively small number of elements in $\mathcal{T}^\psi$ that do not intersect T. In this case, we expect the search region approach to be efficient. However, if T is obtuse, as shown on the right, then the search region, outlined in green, is mostly comprised of elements that do not intersect T. In this case the search region approach will not reduce the number of element intersection tests enough to overcome the computational overhead required to construct the specialized search tree.

We do not necessarily need to guarantee that the search region contains all elements in $\mathcal{T}^\psi$ that intersect T. If needed, we could compute a region in which the intersecting elements are most likely to occur, and then use a supplementary method to find any intersecting elements outside the region. The most likely supplementary method would be a recursive neighbor-checking algorithm. Given an intersecting element on the edge of the search region, we would

check each neighbor that does not lie in the region for element intersection. If that neighbor intersects our test element, then we recursively check all of its neighboring elements.

While this approach would guarantee that all intersecting elements are found, the benefit using the search region would again vary depending on the structure of the meshes. For example, if we used the inscribed circle as a search region, then once again the method will be efficient for elements that are close to equilateral. However, if T is highly obtuse, then the search region may not contain most of the intersecting elements of $\mathcal{T}^\psi$, and so any benefits gained by reducing the number of intersection tests would be overshadowed by the pre-processing costs.

Both the edge-intersection and the search region approaches could be effective on certain mesh types. But due to the additional data structures they require and the computational overhead required to create those structures, they have the potential to perform worse than a brute force approach. We did not wish to make any additional assumptions about mesh regularity or size, so we decided to use a simple recursive approach that does not require additional data structures or pre-processing.

Given an element $T \in \mathcal{T}^\phi$, we choose a vertex v_1 of T and attempt to find an element $\bar{T} \in \mathcal{T}^\psi$ so that $v_1 \in \bar{T}$. If we cannot find such an element, then we choose a different vertex of T and try again. If none of the vertices of T are contained in some vertex of $\bar{T}$ then by the convexity of T, $T \cap \bar{T} = \emptyset$ for all $\bar{T} \in \mathcal{T}^\psi$.

If we have a vertex v contained in an element $\bar{T} \in \mathcal{T}^\psi$, we compute $T \cap \bar{T}$ and then recursively test every neighbor of $\bar{T}$ for intersection with T. In order to avoid redundant intersection tests, we maintain a Boolean array of the elements in $\mathcal{T}^\psi$ we have tested.

This recursive method requires us to efficiently search the mesh $\mathcal{T}^\psi$ for the element that contains a given point. Since we are assuming our elements form a Delaunay triangulation, we can employ an algorithm described by Green and Sibson [8]. In this algorithm, we pick an initial element in the space we are searching over, and consider a line from the center of that element to our given query point. We then walk, from element to element, across that line until reaching the element containing the query point. A analysis by Devroye, Mücke, and Zhu [6], shows this algorithm to have an expected running time of $O(n^{1/3})$ operations.

However, if $\mathcal{T}^\psi$ contains a hole, then we must rely on a more general searching algorithm. Krause and Rank [9] present an algorithm for point-location in quasi-uniform meshes that requires $O(nlog(n))$ operations for preprocessing and $O(log(n))$ operations for each search.

Altogether, we have the following algorithm for computing sets of element intersections:

Algorithm 2 Element Intersection Computation

Require:
$T \in T^{\phi}$

$i \leftarrow 1$
$found \leftarrow false$
while $i < 4$ && $!found$ **do**
  $v \leftarrow i^{th}$ vertex of T
  $\bar{T} \leftarrow$ element of T^{ψ} containing v
  if $\bar{T} \neq \emptyset$ **then**
    $found \leftarrow true$
  $i \leftarrow i + 1$
if $\bar{T} = \emptyset$ **then return** $\emptyset$
return recursive_intersect($\mathcal{T}^{\psi}$, T, $\bar{T}$)

Algorithm 3 Recursive Element Intersection

Require:
$T \in \mathcal{T}^{\phi}$
$\bar{T} \in \mathcal{T}^{\psi}$
$\mathcal{T}$, a structure containing the nodes, connectivity matrix, and adjacency matrix for the mesh $\mathcal{T}^{\psi}$
I, a mapping from elements $\bar{T} \in \mathcal{T}^{\psi}$ to the vertices of $T \cap \bar{T}$
searched, a Boolean array corresponding to the elements in $\mathcal{T}^{\psi}$ that have already been tested for intersection with T

$searched(\bar{T}.index) \leftarrow true$
$int\_vertices \leftarrow compute\_intersect(T, \bar{T})$
if $int\_vertices \neq \emptyset$ **then**
  $I(\bar{T}.index) \leftarrow int\_vertices$
  for $i = 1, 2, 3$ **do**
    $\hat{T} \leftarrow neighbors(\bar{T}, i)$
    if $!searched(\hat{T}.index)$ **then**
      $[I, searched] = recursive\_intersect(\mathcal{T}^{\psi}, T, \hat{T}, I, searched)$
return $[I, searched]$

Chapter 4

Numerical Studies

Theoretically, the orthogonal projection must be an optimal interpolant in the L^2-norm. However, we wish to quantify the effectiveness of the projection algorithm in an iterative mesh refinement scheme compared to simple mesh interpolation.

4.1 Bratu's Equation

To examine the behavior of our projection algorithm, we use it to find solutions to Bratu's nonlinear boundary value. Given a forcing function $f : \mathbb{R}^2 \to \mathbb{R}$, we attempt to find the solution $u : \mathbb{R}^2 \to \mathbb{R}$ to

$$-\Delta u(x,y) - \lambda e^{u(x,y)} = f(x,y) \qquad \text{in } \Omega = (0,1)^2 \tag{4.1}$$

$$u \equiv 0 \qquad \text{on } \partial\Omega \tag{4.2}$$

using an iterative mesh refinement scheme. For simplicity, let $\lambda = 1$ and choose the forcing function $f(x,y) = -2x(x-1) - 2y(y-1) - e^{x(x-1)y(y-1)}$ so that we have a known polynomial solution $u(x,y) = x(x-1)y(y-1)$.

Experiment 1

Before using the projection algorithm in an iterative scheme, we examine the error generated by projecting a finite element solution to (4.1) onto a finer mesh. In this experiment we let $V^k(\Omega)$ be a finite element space constructed over nearly uniform triangular elements of diameter 2^{-k} and define $u^e \in V^k(\Omega)$ such that $u^e(v) = u(v)$ for all nodes v of $V^k(\Omega)$. We then project u^e onto $V^{k+1}(\Omega)$, yielding an approximation u^p, and construct the interpolation

$u^i \in V^{k+1}(\Omega)$ such that $u^i(\bar{v}) = u^e(\bar{v})$ for all nodes $\bar{v}$ of $V^{k+1}(\Omega)$. Finally, we compute the errors $\|u - u^p\|_{L^2}$ and $\left\|u - u^i\right\|_{L^2}$ for $k = 2, 3, 4, 5, 6$.

From the results given in Table 4.1, we can see that the error from the interpolation are four to nine times higher than the error from the projection. Furthermore, if we assume the errors have the form $\|u - \tilde{u}\|_{L^2} = Ch^n$ then we see the error in the interpolation is of first order. The error in the projection does not appear to be strictly polynomial, but we see that is superlinear. So the projection becomes more accurate compared to the interpolation as the mesh size decreases.

Experiment 2

To compare interpolation errors in a more realistic setup, we first use the Galerkin method to find an approximate solution to the nonlinear equation (4.1) on a nearly uniform mesh of diameter 2^{-k}, where $k = 2, 3, 4, 5, 6, 7$. Then, using an adaptive mesh refinement scheme as discussed in Section 2.2, we construct a refinement that attempts to equidistribute the error across all elements and reduces the overall error by one-half. Finally, we compute the projection of the previous result onto the refinement and compare the resulting error to that of the comparable interpolation. The errors computed in this experiment are given in Table 4.2 while the total running time for each refinement is given in Table 4.3.

These results are very similar to those from the first experiment. Again, the error in the interpolation is approximately four times larger than the error in the projection on the coarsest mesh, and the projection is comparatively more accurate as the mesh size decreases. Again we see the interpolation has roughly first order error while the projection shows superlinear error. However, when the mesh diameter reaches 2^{-7}, the large number of floating point operations required to compute the projection matrix causes the round-off error to overcome the theoretical benefits of the orthogonal projection.

In Table 4.3, we also see that the time required to compute the orthogonal projection is 12.6 to 28.7 times longer than the time required to compute the interpolation. This suggests the projection algorithm will only be more time efficient than interpolation when the convergence of Newton's method is strongly dependent on the accuracy of the initial guess.

k	$\left\|u-u^i\right\|_{L^2}$	order	$\left\|u-u^p\right\|_{L^2}$	order
2	1.2927959509e-02	-	3.2063111207e-03	-
3	5.3816384579e-03	1.2643772065	1.5920452746e-03	1.0100330586
4	2.8574440693e-03	0.9133202131	4.6839453479e-04	1.7650852161
5	1.4339158115e-03	0.9947649373	2.0038844581e-04	1.2249249160
6	6.9390591590e-04	1.0471483510	7.0401402347e-05	1.5091232549

Table 4.1: Interpolation vs projection errors on meshes of size $h = 2^{-k}$.

k	$\left\|u-u^i\right\|_{L^2}$	order	$\left\|u-u^p\right\|_{L^2}$	order
2	1.3217557908e-02	-	4.0035391531e-03	-
3	5.5877034506e-03	1.2421282859	1.2504296136e-03	1.6788520644
4	2.6319471031e-03	1.0861249623	3.8932873272e-04	1.6833631253
5	1.3456124525e-03	0.9678675323	1.6707918108e-04	1.2204568430
6	6.4960091245e-04	1.0506373976	4.9770982818e-05	1.7471551964
7	3.2808777128e-04	0.9854718373	5.8596162146e-04	-3.5574293950

Table 4.2: Interpolation vs projection errors after one refinement.

k	2	3	4	5	6	7
Interpolation	0.0962	0.2948	0.8900	7.6441	65.1192	568.4807
Projection	1.6262	7.0645	23.6278	152.4487	823.4011	11527.2298

Table 4.3: Computation time (in seconds) to interpolate from a mesh of size $h = 2^{-k}$ to a mesh refinement that is of size $2^{-(k+1)}$.

Experiment 3

To test the time efficiency of the projection method, we now solve Bratu's boundary value problem using the full adaptive mesh refinement scheme. We solve the problem once using only the interpolation u^i as defined in the first experiment as an initial guess for each refinement, and again using only the projection u^p. The number of Newton iterations required for convergence on each mesh are given below in Table 4.4 while the total time required to compute the solutions are recorded in Table 4.5.

Refinement	1	2	3	4	5	6	7	8
Newton Iterations (Interpolation)	3	3	3	3	3	3	3	3
Newton Iterations (Projection)	3	3	3	2	2	2	2	2

Table 4.4: Iterations required for convergence of Newton's method at each refinement.

Refinement	1	2	3	4	5	6	7	8
Interpolation	0.136	0.225	0.606	1.447	3.710	12.215	36.161	99.442
Projection	1.306	2.976	7.284	17.222	39.979	98.373	263.722	816.974

Table 4.5: Total computation time (in seconds) required for each refinement.

While the projection does save a single Newton iteration in the later refinements, the time saved calculating the solution is dominated by the time required to compute the projection itself. To see a time benefit from using the projection, we will need to apply it towards more difficult problems.

Experiment 4

Consider the same setting as the previous three experiments, but let $\lambda = 6.81$ and

$$f(x,y) = -2c\left(x(x-1) + y(y-1)\right) - \lambda e^{cx(x-1)y(y-1)}$$

which gives a solution of $u(x,y) = cx(x-1)y(y-1)$ for any $c > 0$. As c increases, the Jacobian of the weak residual becomes increasingly ill-conditioned, and so a poor initial guess may cause Newton's method to fail to converge.

We increase c from 50 to 100, and for each value of c, we find a finite element solution using our adaptive mesh refinement scheme with four refinements. We compute a solution using only interpolation and another using only projection, and we record the number of Newton

iterations required for convergence at each refinement. The results of this experiment are given in Figure 4.1.

When both methods converge, the interpolation saves approximately six Newton iterations over four refinements. However, using interpolation, our method begins to fail at $c = 56$, and it fails for all $c \geq 67$. Meanwhile when we use our projection algorithm, we have convergence for almost all values of c up to 75, and for some values of c up to 98.

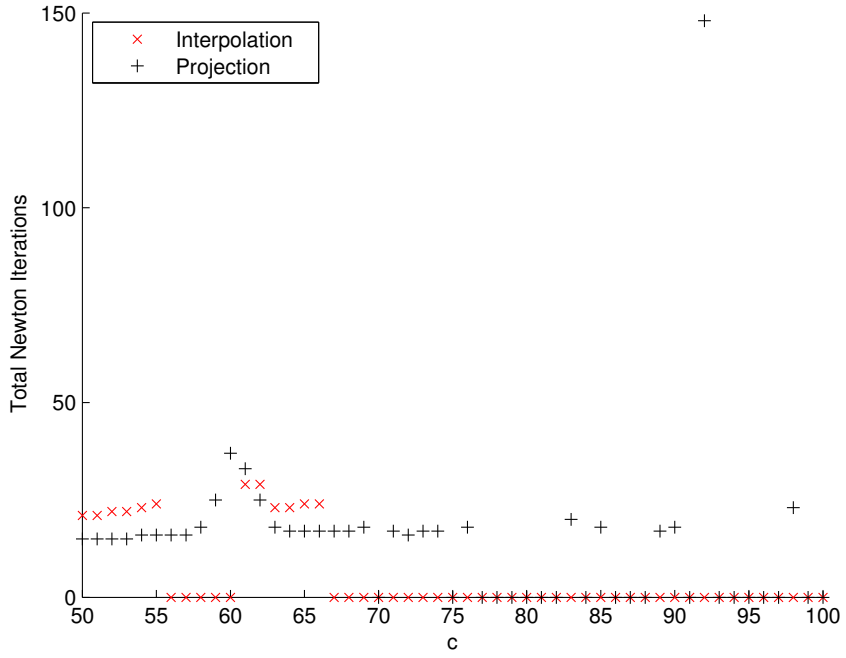

Figure 4.1: Total Newton iterations required to compute a solution to (4.1) with $f = (x, y) = -2c\left(x(x-1) + y(y-1)\right) - e^{cx(x-1)y(y-1)}$ *with c varying from 50 to 100. Zero iterations indicates the method failed to converge.*

4.2 The Bratu-Gelfand Problem

We again attempt to solve Bratu's boundary value problem, but now we set the forcing function to zero. Consider the solution $u : \mathbb{R}^2 \to \mathbb{R}$ that satisfies

$$-\Delta u(x, y) - \lambda e^{u(x,y)} = 0 \qquad \text{in } \Omega = (0, 1)^2 \tag{4.3}$$

$$u \equiv 0 \qquad \text{on } \partial\Omega. \tag{4.4}$$

This is a common test problem for continuation methods for nonlinear elliptic eigenvalue problems. A numerical study by Chan and Keller [4] shows that this problem has two solutions for $\lambda < 6.81$, no solutions for $\lambda > 6.81$ and exactly one solution at $\lambda \approx 6.81$. Tracing solution curves for this eigenvalue problems typically requires specialized methods as nonlinear Galerkin methods fail as λ approaches the critical point 6.81.

We attempt to solve the Bratu-Gelfand problem with λ approaching 6.81 without any modification to the adaptive mesh refinement scheme used in Section 4.1. We begin with $\lambda = 6.8$ and increase the parameter with increments of 0.001. For each value of λ, we compute a solution using eight refinements and record the number of Newton iterations required for convergence. In Table 4.6 we list the total number of Newton iterations used to compute a solution for each value of λ.

Both methods fail for $\lambda > 6.80812$ and they both fail at the same refinement for each value of λ. So while using the projection algorithm saves Newton iterations compared to using interpolation, it does not allow us to compute solutions closer to the critical point $\lambda = 6.81$.

λ	6.8	6.801	6.802	6.803	6.804	6.805	6.806	6.807	6.808
Interpolation	45	45	47	48	49	51	52	56	69
Projection	29	29	30	30	29	29	31	31	35

Table 4.6: Total Newton iterations required to compute a solution to (4.3) with λ ranging from 6.8 to 6.808.

4.3 Solutions Near Singular Points

Again looking to the literature of nonlinear elliptic eigenvalue problems, we consider the boundary value problem

$$-\Delta u(x,y) - \lambda\left(u(1-\sin(u)) + u^k\right) = f(x,y) \qquad \text{in } \Omega = (0,1)^2 \tag{4.5}$$

$$u \equiv 0 \qquad \text{on } \partial\Omega, \tag{4.6}$$

where $\lambda, k > 0$. In the one-dimensional version of this problem with $f \equiv 0$, Wang [12] has shown that when $k = 1$, the bifurcation curve for this problem has infinitely many folds. Although a numerical study by Chang et al. [5] suggests this result does not hold for the two-dimensional problem, we expect that solutions to (4.5) will become increasingly difficult to compute as k approaches 1.

We compute solutions to (4.5) using the forcing function

$$f(x,y) = 2x(1-x) + 2y(1-y) - \lambda\left(w(x,y)(1-\sin(w(x,y))) + w(x,y)^k\right)$$

where $w(x,y) = x(1-x)y(1-y)$ is the solution to (4.5) for this f. Using $\lambda = 10$, we let $k = 1.15, 1.1, 1.05, 1$. For each k, we compute the finite element solution using at least four refinements once using interpolation and again using projection. The number of iterations required for convergence on each refinement and the total time required to compute a solution on each refinement are displayed in Table 4.7 and Table 4.8 respectively.

As k approaches 1, the convergence of Newton's method becomes increasingly sensitive to the initial guess. By using the projection rather than interpolation, we reduce the number of Newton iterations required for convergence by a factor of two on coarser meshes and by a factor of 45 on the finest mesh. From Table 4.8 we see that the computational savings from the reduced number of Newton iterations overcomes the cost of computing the projection matrix for every refinement in this experiment.

Examining the computation time for $k = 1$ more closely, we separate the time required to compute the interpolation and the projection from the time required to compute the finite element solution using Newton's method.

Note that the order of the projection error is superlinear. So in Table 4.10 we see the time required for Newton's method to converge decreases from refinement 2 to refinement 4 even though the size of the linear system solved with each Newton iteration increases quadratically with each refinement. The interpolant sees only a linear reduction in error as the mesh size increases, and so we see the time required for Newton's method to converge roughly doubles with each refinement. So the computation time saved by reducing the number of Newton iterations needed for convergence outpaces the increased time needed to compute the projection.

Refinement	1	2	3	4	5	6
Interpolation, $k = 1.15$	94	88	76	72		
Projection, $k = 1.15$	52	31	30	18		
Interpolation, $k = 1.1$	126	115	105	92		
Projection, $k = 1.1$	63	29	34	20		
Interpolation, $k = 1.05$	192	174	158	139		
Projection, $k = 1.05$	61	37	27	22		
Interpolation, $k = 1$	443	395	355	322	226	227
Projection, $k = 1$	237	174	64	11	8	5

Table 4.7: Iterations required for Newton's method to converge when $\lambda = 10$ and k varies from 1.15 to 1.

Refinement	1	2	3	4	5	6
Interpolation, $k = 1.15$	17.880	25.957	47.044	99.788		
Projection, $k = 1.15$	15.167	21.569	46.977	90.596		
Interpolation, $k = 1.1$	23.962	33.656	65.709	128.798		
Projection, $k = 1.1$	17.548	20.913	49.742	95.655		
Interpolation, $k = 1.05$	25.247	50.118	96.999	189.790		
Projection, $k = 1.05$	13.706	23.042	44.904	97.616		
Interpolation, $k = 1$	51.726	102.993	205.177	418.781	752.725	1437.866
Projection, $k = 1$	33.444	56.930	65.586	82.620	197.927	524.976

Table 4.8: Time (in secs) required to compute the finite element solution on each refinement when $\lambda = 10$ *and* k *varies from* 1.15 *to 1.*

Refinement	1	2	3	4	5	6
Interpolation	0.175	0.413	1.087	3.234	12.091	35.881
Projection	5.571	12.520	28.575	68.013	173.546	485.770

Table 4.9: Time (in secs) required to compute the interpolation and the projection on each refinement when $\lambda = 10$ *and* $k = 1$.

Refinement	1	2	3	4	5	6
Interpolation	51.551	102.580	204.090	415.547	740.634	1401.985
Projection	27.873	44.410	37.008	14.606	24.381	39.206

Table 4.10: Time (in secs) required for Newton's method to converge on each refinement when $\lambda = 10$ *and* $k = 1$.

Chapter 5

Conclusion

We developed an algorithm for projecting solutions between finite element spaces with different element topologies primarily with the goal of finding an accurate initial guess when solving nonlinear problems using Newton's method within mesh refinement schemes. From our numerical results, we found the projection algorithm produces a finite element solution with one-fourth to one-ninth of the error (in the L^2-norm) of a standard interpolant. Furthermore, using the approximate solution produced by our algorithm instead of the interpolant as an initial guess for Newton's method reduced the number of iterations required for convergence in all but the simplest problem studied.

Although finding the projection requires considerable computational cost, our algorithm provided computational savings when used to solve nonlinear problems that challenge standard Newton solvers. Furthermore, the majority of the cost in our algorithm comes from computing the projection matrix, which is only dependent on the finite element spaces acting as the source and target of the projection. In problems that allow for mesh reuse, for example time-varying problems in which approximation error is not strongly dependent on time, the projection matrix may be computed only once and then reused to compute multiple projections between the same two meshes. Since this projection algorithm is readily parallelizable, implementations that take advantage of multicore computer architectures would make adaptive algorithms based on our approach more attractive in a wider range of problems.

In addition, finite element error estimates are often given in terms of best approximation error. The algorithm described in this thesis may be easily modified to compute the H^m orthogonal projection for $m = 0, 1, 2, ...$, producing a finite element solution that is the best approximation in the H^m-norm. With this modification, our algorithm may be of interest in validating theoretical results involving multiple finite element spaces.

Bibliography

[1] E.L. Allgower and K. Böhmer. Application of the mesh independence principle to mesh refinement strategies. *SIAM Journal on Numerical Analysis*, 24(6):1335–1351, 1987.

[2] E.L. Allgower, K. Böhmer, F.A. Potra, and W.C. Rheinboldt. A mesh-independence principle for operator equations and their discretizations. *SIAM Journal on Numerical Analysis*, 23(1):160–169, 1986.

[3] Dietrich Braess. *Finite elements: Theory, fast solvers, and applications in solid mechanics*. Cambridge University Press, 2007.

[4] Tony F.C. Chan and H.B. Keller. Arc-length continuation and multigrid techniques for nonlinear elliptic eigenvalue problems. *SIAM Journal on Scientific and Statistical Computing*, 3(2):173–194, 1982.

[5] S-L. Chang, C-S. Chien, and B-W. Jeng. Tracing the solution surface with folds of a two-parameter system. *International Journal of Bifurcation and Chaos*, 15(08):2689–2700, 2005.

[6] Luc Devroye, Ernst P. Mücke, and Binhai Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22(4):477–482, 1998.

[7] Ronald Goldman. Intersection of two lines in three-space. In *Graphics Gems*, page 304. Academic Press Professional, Inc., 1990.

[8] Peter J. Green and Robin Sibson. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21(2):168–173, 1978.

[9] Roland Krause and Ernst Rank. A fast algorithm for point-location in a finite element mesh. *Computing*, 57(1):49–62, 1996.

[10] Mats G. Larson and Fredrik Bengzon. The finite element method: Theory, implementation, and practice, 2010.

[11] Michael James David Powell. *Approximation theory and methods*. Cambridge University Press, 1981.

[12] Shin-Hwa Wang. On the evolution and qualitative behaviors of bifurcation curves for a boundary value problem. *Nonlinear Analysis: Theory, Methods & Applications*, 67(5):1316–1328, 2007.

[13] O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery (SPR) and adaptive finite element refinement. *Computer Methods in Applied Mechanics and Engineering*, 101(1):207–224, 1992.