To the Graduate Council:
I am submitting herewith a dissertation written by Michael A. Saum entitled "Adaptive Discontinuous Galerkin Finite Element Methods for Second and Fourth Order Elliptic Partial Differential Equations". I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Mathematics.

Ohannes Karakashian
Major Professor

We have read this dissertation
and recommend its acceptance:

Chuck Collins

Jack Dongarra

Xiaobing Feng

Accepted for the Council:

Anne Mayhew
Vice Chancellor and Dean of
Graduate Studies

(Original signatures are on file with offical student records.)

Adaptive Discontinuous Galerkin Finite Element Methods for Second and Fourth Order Elliptic Partial Differential Equations

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Michael A. Saum
August 2006

Dedication

To the memory of my father, James Arthur Saum, who always supported me in whatever I wanted to do, and my mother, Eunice Saum, who continues to support me while I pursue my dreams.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Professor Ohannes Karakashian, who has provided the inspiration for this work. In addition, he has provided the expert guidance which has kept me focused on the problems at hand, and if it were not for his patience and genuine interest in seeing me achieve my goals, I would not have gotten this far. Finally, his amazing knack for identifying computer "bugs" saved me countless hours when I could not see the forest through the trees.

I would also like to thank my Dissertation Committee, Dr. Xiaobing Feng, Dr. Chuck Collins, and Dr. Jack Dongarra who have always had time for me.

Lastly, I would like to thank my family. Without their help, encouragement, and love I would not have embarked on the journey that has become my "second career".

Abstract

A unified mathematical and computational framework for implementation of an adaptive discontinuous Galerkin (DG) finite element method (FEM) is developed using the symmetric interior penalty formulation to obtain numerical approximations to solutions of second and fourth order elliptic partial differential equations. The DG-FEM formulation implemented allows for h-adaptivity and has the capability to work with linear, quadratic, cubic, and quartic polynomials on triangular elements in two dimensions. Two different formulations of DG are implemented based on how fluxes are represented on interior edges and comparisons are made. Explicit representations of two a posteriori error estimators, a residual based type and a "local" based type, are extended to include both Dirichlet and Neumann type boundary conditions on bounded domains. New list-based approaches to data management in an adaptive computational environment are introduced in an effort to utilize computational resources in an efficient and flexible manner.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Mathematical modeling using partial differential equations (PDEs) is an important focus of the scientific community, from the very small scale of atomic and molecular phenomena to the very large scale of supernovae explosions. Continuous advances of high end computing systems implies that a continuous demand for updates in the algorithms, software, and tools that will effectively use those new systems must also be developed. There are two fundamental problems related to numerical approximation of PDEs, determining the accuracy and reliability of the methods involved and utilizing the computational resources efficiently and effectively. Adaptive numerical methods attempt to tailor the amount of work involved to obtain approximations to the problem being solved, say, having a finer mesh where the solution varies rapidly and coarser mesh where the solution does not vary as much. Adaptive methods are by their nature complex to implement, yet they have the potential to reduce drastically the size of the problems being solved. In addition, adaptive methods are notorious for their computational overhead which often becomes comparable to the possible gains from the approach.

This dissertation addresses implementation of adaptive methods for a specific variant of the the Finite Element Method (FEM)[1] called the Discontinuous Galerkin Method (DG). Briefly, the DG method differs from the standard Galerkin FEM in that continuity constraints are not imposed on quantities on the interelement boundaries, thus resulting in a solution which is composed of totally piecewise discontinuous functions.

This dissertation contains a mix of mathematical theory and computer science implementation detail, all directed toward obtaining efficient adaptive approximations to "representative" second and fourth order elliptic PDEs utilizing the DG FEM method.

1.1 Background

Adaptive methods for PDEs refer to methods where the approximate solution of the PDE is obtained with a prescribed accuracy in an automatic adaptive process of modifying the computational mesh to the local behavior of the solution. The mesh adaptation process is driven by a posteriori errors estimated for regions of the domain. For an excellent overall treatment of a posteriori error estimation, the reader is referred to Verfürth (1995); Babuşka and Strouboulis (2001).

[1] For more information on the standard FEM, the reader is referred to the excellent treatises Ciarlet (1978), Johnson (1992), and Brenner and Scott (2004) among others.

Most would agree that a posteriori error estimation for elliptic problems and subsequently for other PDEs begins with the pioneering paper of Babuşka and Rheinboldt (1978) and continues with studies dedicated to what can be classified as a *Residual Based* method (Verfürth, 1998). In this approach local residuals are calculated and then the a posteriori error estimator is obtained by solving local Dirichlet or Neumann problems, taking the residuals as data (Babuşka and Rheinboldt, 1978; Bank and Smith, 1993). Another approach of the method uses Galerkin orthogonality, a priori interpolation estimates, and global stability in order to get error estimators in global L^2- and H^1-norms (Eriksson and Johnson, 1998).

Solving finite element problems in an enriched function space (by hierarchical bases) produces the so called *Hierarchical Based* error estimators (Bank and Smith, 1993). There also exist error estimators that attempt to control the error or its gradient in the maximum norm. Such estimators are based on optimal a priori estimates for the error in the maximum norm (Eriksson and Johnson, 1998).

The Discontinuous Galerkin (DG) can be traced back to the paper of Reed and Hill (1973). It is important to note that it is not a single method but rather a methodology which has at its core that it uses approximating functions that have no continuity constraints imposed on interelement boundaries.

In recent years there has been increased interest in the DG method, much of it with regard to convection dominated flows (Cockburn, 1997; Cockburn and Shu, 1998). The absence of continuity constraints on the interelement boundaries implies that one has a great deal of flexibility to the method, at the cost of increasing the number of degrees of freedom. This flexibility is the source of many but not all of the advantages of the DG method over the Continuous Galerkin (CG) method that uses spaces of continuous piecewise polynomial functions and other "less standard" methods such as nonconforming methods. One great advantage lies in the fact that one is able to easily refine or coarsen the mesh locally or to vary the degree of the piecewise polynomials across the mesh.

The version of the DG method that we have pursued is a weak formulation that includes penalty terms on the jumps of the functions over the interelement boundaries. For this reason, it is referred to as an "interior penalty" method, the one we use here in this research is often termed the "symmetric interior penalty" or SIPG method. This approach can be traced back to the paper of Nitsche (1971). Since then the SIPG has been a major area of interest for a number of researchers including Douglas Jr. and Dupont (1976), Baker (1977), Wheeler (1978), Arnold (1982) and Baker et al. (1990); Karakashian and Jureidini (1998). There are other variants of the DG method including the cell-discretization method of Greenstadt (1982), and recently, a skew symmetric formulation in which the penalty terms are removed (Baumann, 1997; Baumann and Oden, 1999; Rivière et al., 1999) Unlike the SIPG, these variants do not lead to symmetric matrices upon discretization; having symmetric systems to solve implies that standard linear solvers such as conjugate gradient type methods can be utilized. For a nice survey of DG methods see Cockburn et al. (2000); Arnold et al. (2002); Cockburn and Shu, editors (2005). There exists even another variant for second-order elliptic problems which is known as the Local Discontinuous Galerkin (LDG) method, (Arnold et al., 2002), which is basically a mixed-DG formulation and it yields a non-symmetric stiffness matrix. It is interesting to note that throughout all of these variants of DG methods, there are some similarities. For example, LDG method also contains "jump" terms which can be viewed as penalty terms.

1.2 Contributions

First of all, this research extends the results of the two papers, Karakashian and Pascal (2003) and Karakashian and Pascal (2004) in a number of areas. Both of these papers dealt with SIPG formulation for second order elliptic partial differential equations (PDEs), we extend the formulations to include fourth order elliptic PDEs, primarily the biharmonic problem. Karakashian and Pascal (2003) introduced the concept of solving a "local problem" which can be applied as a correction to the solution and thus also can be used as an a posteriori error estimator. The formulation presented there used homogeneous Dirichlet boundary conditions. We extend the concepts presented there to obtain explicit expressions for problems with non-homogeneous Dirichlet and Neumann boundary conditions on different parts of the domain boundary. In both papers, mention is made that the Arnold and the Baker formulations for representation of fluxes on interior edges are basically equivalent. We illustrate and compare the numerical differences between these methods.

For the fourth order elliptic PDE, we introduce a heuristic residual-based type error estimator and obtain an explicit formulation of a local problem based error estimator.

Our ideas put forth on how to organize and manage data and calculations in an adaptive scientific computing environment provide a good foundation which can be built and improved upon. We integrate state of the art computer software with our algorithms to produce efficient code. We also identify areas where improvements can be made and implement various "proof of concept" techniques to also build upon obtaining better performance.

Finally, last but certainly not least, is the actual program development. The computer implementations for this research consists of around 28,000 lines of `C` code; 13,000 for the second order elliptic `Ell2` implementation and 15,000 for the fourth order elliptic `Ell4` implementation. In actuality though, only approximately 18,000 lines of "unique" `C` code was developed, since modular design principles allowed for extensive code reuse between `Ell2` and `Ell4`. This coding effort will provide a solid foundation on which one can explore the DG method applied to two important PDEs. This source code will be available in electronic form in some manner in the near future, and is not included in the text of this dissertation. Please feel free to contact the author to inquire about source code availability.

1.3 Thesis Organization

This dissertation is divided into three main parts. The first part, Chapter 2, describes computer science and computational aspects involved in the programming implementation of our DG-FEM implementation. It contains information on wide variety of subjects, from data organization and layout to quadrature to marking strategies to linear solver implementations such as Multigrid and preconditioned conjugate gradient. Finally, various optimization techniques that were employed are identified and explained.

The second part, Chapter 3, describes the mathematics of the DG formulation for second order elliptic PDEs. Included in this chapter is a review of the two a posteriori estimators utilized. Note that Appendix A contains a detailed description of stiffness matrix assembly for the model problem. This chapter also contains a numerical comparison between the Arnold and the Baker formulations, including representative meshes for a variety of interesting test problems. Finally, a brief numerical investigation as to how sensitive the results are to the choice of the penalty parameter γ is provided. Appendix B provides a listing of the test problems used in the investigations.

The third part, Chapter 4, describes the mathematics of the DG formulation for fourth order elliptic PDEs, specifically the biharmonic problem. This chapter basically reproduces the structure

of Chapter 3, however a penalty parameter sensitivity study is not included. Appendix C provides more detail on stiffness matrix assembly, and Appendix D identifies the test problems used during the investigations.

I close by summarizing some of the more interesting results and attempt to identify future areas of research which could prove even more fruitful.

Chapter 2

Computational Aspects

This chapter provides the basic information required to describe the implementation of the computational models described in later chapters. In some respects, this chapter is a repository of topics which do not fit cleanly into other chapters, but which are referenced throughout the rest of this thesis.

2.1 Program Organization

For the most part, all programs written in support of this thesis are written in the `C` programming language[1]. One of the goals of the research associated with this thesis was to develop methods and techniques which allow the programs to run as fast as possible on a wide variety of hardware and software platforms. From a flexibility standpoint, `C` is definitely the language of choice here from the ease of dynamically allocating memory to the efficient use of pointers. From a portability standpoint, `C` also has an advantage (note that all commercial Linux workstations come with the `gcc` compiler system). Finally, linking software with other packages already in existence to perform specific tasks (see §2.11) is usually more easily done with programs written in `C` than from programs written in `FORTRAN` or `C++`. While there are exceptions to these reasons, the author has decided to avoid the overhead and sometimes obtuse nature of coding in `C++` and to avoid the complexity of maintaining many specialty array indices associated with coding in `FORTRAN`. Thus, `C` is the programming language of choice.

2.1.1 Design Philosophy.

Design and implementation of the Finite Element Method (FEM) into computer programs involves many different data types and algorithms which must all work together. This necessitates a well thought out plan on the *bookkeeping* aspects in contrast to the *computational* aspects of the code. The reader is encouraged to keep in mind the following design goals which have driven the actual writing the FEM code:

- Modular program design,
- Separation of geometric mesh data from PDE data, and,
- Optimize for performance.

[1] The definitive reference for `C` programming is Kernighan and Ritchie (1978).

The modular design allows for reusing portions of code for further experiments. For example, approximately seventy five percent of the program developed for second order elliptic PDEs was reused without change in the biharmonic PDE implementation, specifically the mesh refinement code and linear solver code. The separation of the geometric mesh data from the PDE data allowed this to occur rather easily. Optimizing for performance occurred throughout the development process, and specific areas in which this design goal was addressed will be noted throughout the text.

2.1.2 Components

It is useful at this point to describe the main program components used in the implementation of the DG FEM. `Ell2` is the code which works with second order elliptic PDEs (see §3). `Ell4` is the code which works with the biharmonic PDE. Both implementation follow the same main algorithm, which is illustrated in Algorithm 1. Table 2.1 identifies the main routines and their purpose. These routines will be referred to on occasion throughout the rest of this document. Note that Algorithm 1 describes an adaptive scheme which will continue until either the mesh has reached a specified input level $L_{\max}$, a specified input maximum number of adaptive iterations $aIter_{\max}$, or there are no triangles marked for refinement (i.e., the specified input adaptive tolerance h_{tol} has been achieved.)

2.2 Mesh Representation

Let $\Omega \subset \mathbb{R}^2$. Suppose we are given a quasiuniform triangular mesh which covers Ω. This mesh is a *simplicial complex* [2] composed of triangles, edges (internal and boundary), and vertices. Suppose this mesh consists of four interrelated sets:

1. $\mathbf{T} = \{K_i\}_{i=1}^{n^T}$, K a *triangle*.
2. $\mathbf{E}^I = \{e_i^I\}_{i=1}^{n^I}$, e^I an *interior edge*.
3. $\mathbf{E}^B = \{e_i^B\}_{i=1}^{n^B}$, e^B a *boundary edge*.
4. $\mathbf{N} = \{v_i\}_{i=1}^{n^V}$, v a *vertex*.

where n^T, n^I, n^B, n^V corresponds to the number of triangles, interior edges, boundary edges, and vertices in each set, respectively. It is important to realize that first, elements of each of these sets are ordered within each set. Second, there exists well defined relations between these sets. For example, each edge e is defined by two vertices $\{v_1, v_2\}$, i.e., its endpoints. Each triangle consists of three edges $\{e_1, e_2, e_3\}$ (and thus three vertices $\{v_1, v_2, v_3\}$.) There are other relations which one can put to good use, these will be discussed in more detail later. Third, one often will talk about a triangular mesh $\mathcal{T} \subset \mathbf{T}$, which is simply a collection of triangles. Because of the above relations (and others), there should be no confusion when talking about $\mathbf{T}$ or $(\mathbf{T}, \mathbf{E}^I, \mathbf{E}^B, \mathbf{N})$, i.e., they are equivalent and can (and will) be used interchangeably throughout this document, being specific

[2] $|a_0 a_1 \dots a_n|$ is called an n-simplex with vertices $a_0, a_1, \dots a_n$ where

$$|a_0 a_1 \dots a_n| = \{\lambda_0 a_0 + \lambda_1 a_1 + \cdots + \lambda_n a_n | \lambda_0 + \lambda_1 + \cdots + \lambda_n = 1, \lambda_i \geq 0\}.$$

One can attach to $\lambda_0 a_0 + \lambda_1 a_1 + \cdots + \lambda_n a_n$ the *barycentric coordinates* $(\lambda_0, \lambda_1, \dots, \lambda_n)$. If $|a_0 a_1 \dots a_n|$ is an n-simplex, every subset $\{a_{i_0}, a_{i_1}, \dots, a_{i_q}\}$ is independent. A q-simplex $|a_{i_0} a_{i_1} \dots a_{i_q}|$ is called a q-face of $|a_0 a_1 \dots a_n|$. In $\mathbb{R}^n$, a 0-simplex $|a_0|$ is a point, a 1-simplex $|a_0 a_1|$ is a line segment, and a 2-simplex $|a_0 a_1 a_2|$ is a triangle. A set $\mathcal{R}$ of simplexes in a Euclidean space $\mathbb{R}^n$ is called a *simplicial complex in* $\mathbb{R}^n$ if $\mathcal{R}$ satisfies three conditions: (i) Every face of a simplex belonging to $\mathcal{R}$ is also an element of $\mathcal{R}$. (ii) The intersection of two simplexes belonging to $\mathcal{R}$ is either empty or a face of each of them. (iii) Each point of a simplex belonging to $\mathcal{R}$ has a neighborhood in $\mathbb{R}^n$ that intersects only a finite number of simplexes belonging to $\mathcal{R}$.

Algorithm 1 Adaptive FEM

```
Read_Inp()
Templs()
Initds()
while L ≤ L_max and aIter ≤ aIter_max do
  Stiff()
  Rhs()
  Solver()
  Apost(·)
  if L < L_max then
    Mark_Tri()
    if Elements marked for refinement then
      Process_Refine_List()
      Refine_Tri_List()
    else
      break NRT = 0
    end if
  else
    break L = L_max
  end if
  aIter ++
end while
```

Table 2.1: Main FEM Implementation Routines

Routine	Description
`Read_Inp()`	This routine reads the input which controls the run.
`Templs()`	This routine sets up quadrature and penalty templates, as well as projection and embedding operators for multilevel solvers.
`Initds()`	This routine initializes most dynamic memory required for the run.
`Stiff()`	This routine assembles the stiffness matrix components.
`Rhs()`	This routine assembles the right hand side of the linear system
`Solver()`	This routine implements the solver chosen.
`Apost(·)`	This routine calculates a posteriori error estimators
`Mark_Tri()`	This routine determines which elements should be marked for refinement based on the a posteriori estimators.
`Process_Refine_List()`	This routine preprocesses the elements marked for refinement and ensures that the mesh does not violate the *two neighbor* condition.
`Refine_Tri_List()`	This routine refines the elements marked for refinement and produces the new mesh.

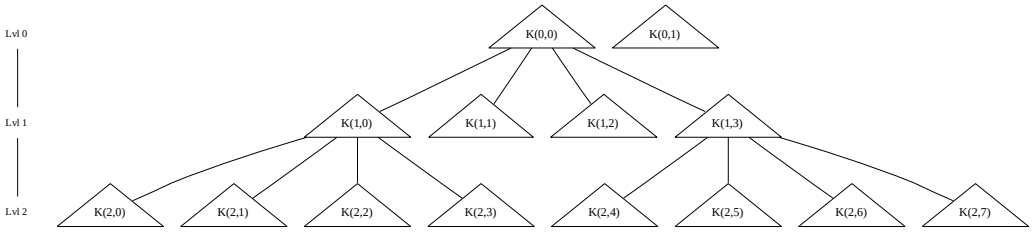


Figure 2.1: Representative Tree $\mathbb{T}$ of depth 2

when necessary. Finally, one should always keep in mind that the complete collection of triangles is $\mathbf{T}$, even though various subsets $\mathcal{T}$ of $\mathbf{T}$ are used in the calculation process at any one time.

In practice, one requires more than simple ordered sets to support the various algorithms utilized in performing a posteriori estimation driven *Adaptive Mesh Refinement (AMR)*. Because any AMR process produces a sequence of collections $\{\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \dots\}$, one requires more information to describe the *state* of a mesh. We have identified the following objectives as being fundamental to our implementation of AMR:

1. Identifying relations (including hierarchy) between elements of $\mathbf{T}$ (if one exists),
2. Identifying criteria used in partitioning strategies applied to $\mathbf{T}$, and
3. Identifying elements of $\mathbf{T}$ which are *active* versus those that are *inactive*.

It is important to realize that this last task is actually part of the second task, however it is so important that it needs to be explicitly mentioned and discussed.

2.2.1 The Tree $\mathbb{T}$

Let us start off by assuming that we are provided with an initial mesh $\mathcal{T}_0$ or $(\mathcal{T}_0, \mathcal{E}_0^I, \mathcal{E}_0^B, \mathcal{N}_0)$. Let each element (triangle) of $\mathcal{T}_0$ become the root node of its own graph. We call this collection $\mathbb{T}$. Note that there is no need to index $\mathbb{T}$ as $\mathbb{T}_0$ since we consider the whole collection of elements as a dynamic data structure which grows and shrinks during the AMR process.

Triangles are refined by regular subdivision, i.e., four *children* triangles are created by connecting the midpoints of the three edges of the *parent* triangle resulting in all children triangles being similar to the parent triangle. The mesh is thus represented by a 4-ary initial mesh rooted tree $\mathbb{T}$ which changes dynamically as coarsening and refinement operations are performed in the AMR process. It is important to note that $\mathbb{T}$ represents only the parent-child relation between elements of $\mathbf{T}$. It does not define explicitly the interrelationships between elements via shared edges, i.e., neighbors. Note also that the tree $\mathbb{T}$ is equivalent to $\mathbf{T}$ because there is a 1-1 correspondence between elements of $\mathbf{T}$ and the nodes of the tree (graph) $\mathbb{T}$. A *representative* tree is shown in Figure 2.1.

Associated with each triangle in $\mathbb{T}$ is the concept of *level*. A triangle's level is simply the depth [3] of that triangle relative to its ancestor at level 0. This is illustrated in Figure 2.1 by the fact that all triangles of the same level are shown on the same horizontal slice of the tree. The level concept is fundamental to object storage and implementing efficient algorithms for AMR. Another way of looking at the level corresponding with a triangle is that there exists a function LEVEL which associates with a triangle an integer level value. The LEVEL function will prove to be useful in future discussions, and so we formally define it here:

[3] A tree node of depth ℓ implies that there exists a direct path consisting of ℓ tree edges connecting the tree node to its ancestor at the root (level = 0) of the tree.

Definition 2.2.1. The triangle tree $\mathbb{T}$ consists of a single (NULL) root node, which has as immediate descendants the initial mesh $\mathcal{T}_0$. Each of these descendants is the root of a 4-ary tree. The leaves of this tree satisfy the following:

$$\text{LEAF}(\mathbb{T}) = \mathcal{T}$$

where **LEAF** is a function which extracts the leaves (nodes with no descendants) from a tree. In addition, one can define a function **LEVEL** which determines a triangle's depth in the tree relative to the root node. Thus, $\text{LEVEL}(K) = 0, \forall K \in \mathcal{T}_0$. Note also that since the root node of the tree is NULL, it has no depth and can be effectively considered to be at the same depth as $\mathcal{T}_0$, i.e., level calculations are from the level corresponding to $\mathcal{T}_0$.

Another useful definition is that of the *height* of a mesh tree:

Definition 2.2.2. For any triangle tree $\mathbb{T}$, the height of $\mathbb{T}$, $\text{HEIGHT}(\mathbb{T})$, is defined as

$$\text{HEIGHT}(\mathbb{T}) = \max_{K \in \mathbb{T}} \text{LEVEL}(K).$$

One can then naturally partition the tree $\mathbb{T}$ into a collection of **level based meshes** or **Composite Level Mesh (CLM)** by

$$\mathbb{T}_s = \{K \in \mathbb{T} | \text{LEVEL}(K) \leq s\}, \quad s = 0, 1, \ldots, \text{HEIGHT}(\mathbb{T})$$

and

$$\mathcal{T}_s = \text{LEAF}(\mathbb{T}_s), \quad s = 0, 1, \ldots, \text{HEIGHT}(\mathbb{T}).$$

It follows that one can refer to a tree $\mathbb{T}$ of depth L where

$$L = \max_{K \in \mathbb{T}} \text{LEVEL}(K) = \text{HEIGHT}(\mathbb{T}).$$

It is important to note that corresponding to the sequence $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_j$ there is also a sequence of trees $\mathbb{T}_0, \mathbb{T}_1, \ldots, \mathbb{T}_j$. Adopting the convention that the tree will only grow from the initial mesh implies $\mathbb{T}_0 \subsetneq \mathbb{T}_1 \subsetneq \cdots \subsetneq \mathbb{T}_j$, and thus we only have to worry about the current tree $\mathbb{T}$ with the current mesh represented as $\text{LEAF}(\mathbb{T}_j) = \mathcal{T}_j$.

2.2.2 Level Based Partitioning

The LEVEL function defined above leads one to define a *level based partition* of tree $\mathbb{T}$ whereby

$$\mathbf{T} = \{T_\ell\}_{\ell=0}^{L}$$

where $T_\ell := \{K \in \mathbb{T} | \text{LEVEL}(K) = \ell\}$. In other words T_ℓ is just the set of triangles which are all of the same level. Since we are starting with a quasiuniform mesh under regular refinement, all triangles of the same level are approximately the same size.

Each set of triangles T_ℓ can further be partitioned into what we will call *Leaf* and *NonLeaf* [4] triangles. A Leaf triangle $K \in \mathbb{T}$ is one that does not have any children, while a NonLeaf triangle $K \in \mathbb{T}$ is a triangle which has been refined and thus does have children. Formally:

$$T_\ell^{Lf} := \{K \in \mathbb{T} | \text{LEVEL}(K) = \ell \text{ and } K \rightarrow \text{children} = NULL\}$$
$$T_\ell^{NLf} := \{K \in \mathbb{T} | \text{LEVEL}(K) = \ell \text{ and } K \rightarrow \text{children} \neq NULL\}$$

[4] Other researchers use the terminology *Active* for Leaf and *NonActive* for NonLeaf.

It thus makes sense to partition each set of triangles T_ℓ as

$$T_\ell = T_\ell^{Lf} \oplus T_\ell^{NLf}, \quad \ell = 0, 1, \ldots, L$$

2.2.3 Mesh Hierarchy $\{\mathcal{T}_\ell\}_{\ell=0}^{L}$

So far we have just been describing different collections of triangles partitioned by level and Leaf and NonLeaf criteria. What is most important however is the concept of a level based mesh $\mathcal{T}_\ell$. Every tree $\mathbb{T}$ induces a collection of level based meshes $\{\mathcal{T}_\ell\}_{\ell=0}^{L}$ as follows:

$$\mathcal{T}_\ell = \bigoplus_{k=0}^{\ell-1} T_k^{Lf} \oplus T_\ell = \bigoplus_{k=0}^{\ell} T_k^{Lf} \oplus T_\ell^{NLf} \tag{2.1}$$

Another way of viewing the level based mesh collections is to take a horizontal cut across the tree $\mathbb{T}$ at a level ℓ, dropping any triangles of level $k > \ell$ which may exist in the tree $\mathbb{T}$. This process could be termed *pruning* the tree $\mathbb{T}$ at level ℓ. Figure 2.2 illustrates a representative mesh hierarchy corresponding to the tree $\mathbb{T}$ illustrated in Figure 2.1.

A couple of comments need to be made regarding $\{\mathcal{T}_\ell\}$. First, the full Leaf $\mathcal{T}_L$ is what is used in Conjugate Gradient and Gauss-Seidel linear solvers, i.e., there are no multilevel meshes required. It is only through Multigrid that the full set of level based meshes are utilized. Second, for any tree $\mathbb{T}$ of depth L, the following relationships exist:

$$\begin{aligned} \mathcal{T}_0 &= T_0^{Lf} \oplus T_0^{NLf} = T_0 \\ T_L^{NLf} = \emptyset \Rightarrow \mathcal{T} := \mathcal{T}_L &= \bigoplus_{\ell=0}^{L-1} T_\ell^{Lf} \oplus T_L \\ &= \bigoplus_{\ell=0}^{L-1} T_\ell^{Lf} \oplus T_L^{Lf} = \bigoplus_{\ell=0}^{L} T_\ell^{Lf} \end{aligned}$$

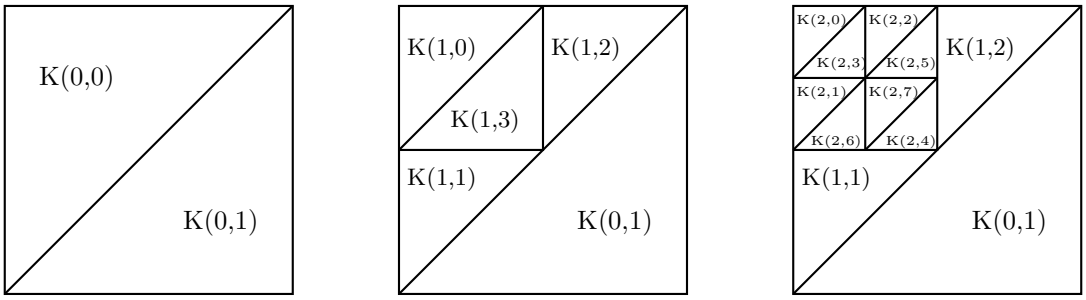
Table 2.2 enumerates the different sets discussed so far for the representative tree $\mathbb{T}$ shown in Figures 2.1 and 2.2. In addition, it should be noted that similar relations exist for interior edges, boundary edges, and vertices contained in the simplicial complex. Finally, the collection $\{\mathcal{T}_\ell\}_{\ell=0}^{L}$ provide the *iterator lists* used throughout computer implementation of the various algorithms.

2.2.4 Local Ordering

Local ordering of vertices and edges is important in any FEM implementation. Figure 2.3(a) illustrates the vertex and edge ordering for an arbitrary triangle K. Note that vertices are numbered in a counterclockwise fashion. Figure 2.3(a) also indicates the local ordering of edges within a triangle. Note here that edge e_i has as its endpoints vertices i and $i/3 \mod 3$.

Since DG allows for hanging nodes along edges, it will often be necessary to perform numerical quadrature along a partial or half edge. Figure 2.3(a) also indicates an ordering scheme for partial edges. Note that the nomenclature of a partial edge e_{ij} indicates that the partial edge is a part of edge e_i in the original triangle K.

Figure 2.3(b) indicates the ordering of the children of K when K undergoes regular refinement to produce $\{K_0, K_1, K_2, K_3\}$. This ordering has some advantages which will be explained in more

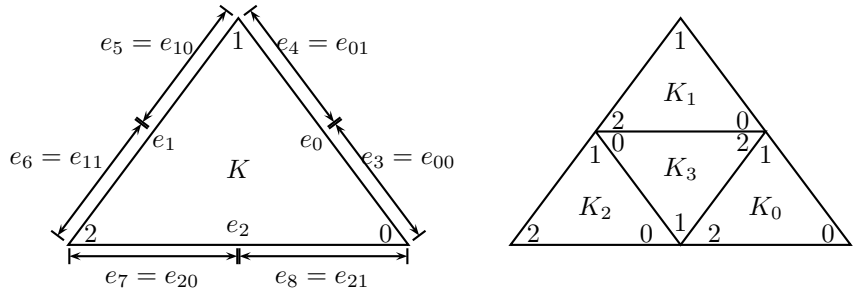


(a) $\mathcal{T}_0$ (b) $\mathcal{T}_1$ (c) $\mathcal{T}_2 = \mathcal{T}_L$

Figure 2.2: Mesh hierarchy $\mathcal{T}$ associated with $\mathbb{T}$ of depth 2

Table 2.2: Enumeration of element collections associated with $\mathbb{T}$ of depth 2

Collection	Representation	Contents
KTree	$\lvert\mathbb{T}\rvert = \lvert\mathbf{T}\rvert$	$\mathbb{T}$ and $\mathbf{T}$ contain the same number of triangles
Level Based Lists	$\mathbf{T} = \{T_\ell\}_{\ell=0}^2$	$T_0 = \{K(0,0), K(0,1)\}$ $T_1 = \{K(1,0), K(1,1), K(1,2), K(1,3)\}$ $T_2 = \{K(2,0), K(2,1), K(2,2), K(2,3),$ $K(2,4), K(2,5), K(2,6), K(2,7)\}$
Level Based Leaf Lists	$\mathbf{T}^{Lf} = \{T_\ell^{Lf}\}_{\ell=0}^2$	$T_0^{Lf} = \{K(0,1)\}$ $T_1^{Lf} = \{K(1,1), K(1,2)\}$ $T_2^{Lf} = \{K(2,0), K(2,1), K(2,2), K(2,3),$ $K(2,4), K(2,5), K(2,6), K(2,7)\}$
Level Based NonLeaf Lists	$\mathbf{T}^{NLf} = \{T_\ell^{NLf}\}_{\ell=0}^2$	$T_0^{NLf} = \{K(0,0)\}$ $T_1^{NLf} = \{K(1,0), K(1,3)\}$ $T_2^{NLf} = \emptyset$
Level Based Meshes	$\{\mathcal{T}_\ell\}_{\ell=0}^2$	$\mathcal{T}_0 = \{K(0,0), K(0,1)\}$ $\mathcal{T}_1 = \{K(0,1), K(1,0), K(1,1), K(1,2), K(1,3)\}$ $\mathcal{T}_2 = \{K(0,1), K(1,1), K(1,2), K(2,0), K(2,1),$ $K(2,2), K(2,3), K(2,4), K(2,5), K(2,6), K(2,7)\}$



(a) K (b) $\{K_0, K_1, K_2, K_3\}$

Figure 2.3: Local Ordering for Triangle K,$\{K_0, K_1, K_2, K_3\}$

detail in §2.3. Suffice it to say that propagation of the outward normals of K to $\{K_0, K_1, K_2, K_3\}$ does not require any additional computation other than a possible change of sign. This means that once the outward normals are computed on the initial mesh $\mathcal{T}_0$, no further normal calculation is required.

2.3 Data Objects

Data objects are the fundamental building blocks of any computer program. The `C` programming language allows for one to group together data of various types into what is called a data structure or em struct. The individual data types of which the struct is composed are often called *fields* of the struct, borrowing from standard database terminology. For our FEM implementation, we have developed structs *types* which provides the basic information required to define the mesh $\mathcal{T}_h$ upon which any FEM is based. In addition, we have separated the data objects into two categories; geometric mesh data and PDE data. This separation provides advantages in reuseability, portability, and is especially suited for DG FEM implementations.

Regarding geometric mesh data objects, it is important to try and keep to a minimum the size (in bytes) of the data object. The main reason is to conserve on storage, because typically FEM implementations require a large number of data objects.

2.3.1 NODE_t Data Objects

NODE_t geometric data objects simply describe the coordinates of the vertices of the triangulation. In addition, we maintain an integer describing the *level* of the vertex. It should be noted that vertices once introduced into the mesh on level ℓ are also present in all future composite level meshes of higher levels, i.e., for levels $\ell' \geq \ell$. Thus, the meaning of the `lvl` field in the NODE_t struct data type is to indicate the *first* level in which the vertex was introduced. The `C` code for the NODE_t struct data type is shown in Figure 2.4. Each instance of the NODE_t data type occupies 24 bytes.

```
/* Node data structure */
typedef struct nodestruct {
  double x;                       // x coordinate
  double y;                       // y coordinate
  unsigned int lvl;               // Level index Node
}
NODE_t;
```

Figure 2.4: NODE_t Structure

2.3.2 EDGE_t Data Objects

There are two types of *EDGE_t* geometric data objects, interior edges (IE) and boundary edges (BE). The `C` code for the EDGE_t struct data type is shown in Figure 2.5. Each instance of the EDGE_t data type occupies 32 bytes. Note that extensive use is made of pointers to other data objects of various types.

Internal Edges

An internal edge can be considered to be the boundary separating all or part of an element from another element. Thus, for each edge present in the mesh, the EDGE_t data struct maintains pointers to TRIANGLE_t data structs, K^+ and K^-. In addition, the position of the edge in both K^+ and K^- is maintained according to the convention described in Figure 2.3(a) in fields `Kploc` and `Kmloc`, respectively. Note that these fields are compressed into unsigned bit fields, thus saving on storage. The presence of the `midpt` field (not NULL) allows for a quick check to see if an edge has been refined.

The `data0` field provides a pointer to each edge's ancestor in the initial mesh, the data struct corresponding to this object is described in Figure 2.6. This field, coupled with the `normlsgn` field and the `lvl` field allows for quick computation of the actual length and normal for this edge. Note that since each edge when refined is split into two edges of equal length, the actual length of an edge on level ℓ is $1/2^\ell$ times the length of its original ancestor edge in the initial mesh. Figure 2.7 illustrates the relationships between normals of the edges of the parent element K and its children $\{K_0, K_1, K_2, K_3\}$. Note that the ordering implemented here implies that the outward normals will only change sign for the middle triangle K_3 when refined.

Boundary Edges

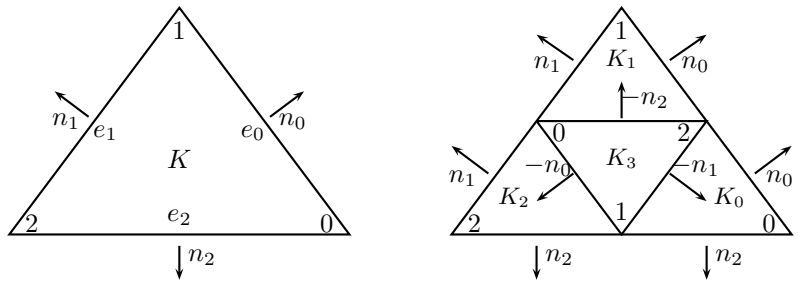
Boundary edges utilize the same struct as interior edges. However, since the edge exists on the boundary, there is no K^- and it is set to NULL. While not all of the fields are used here resulting in under utilized memory, the ratio of boundary edges to interior edges is small for large meshes and thus can be considered to be negligible.

```
/* Edge data structure */
typedef struct edgestruct {
  struct edggeomdata *data0;  // Struct containing edgelen, norml for init mesh
  struct tristruct *Kplus;    // K+
  struct tristruct *Kminus;   // K-
  NODE_t *endp[2];            // Endpoints of edge
  NODE_t *midpt;              // Midpoint NODE (NULL if Leaf EDGE)
  unsigned Kploc : 4;         // Location in K+ (0-8)
  unsigned Kmloc : 4;         // Location in K- (0-8)
  unsigned type : 2;          // Edge Type : 0 - Interior, 1 - Diri, 2 - Neumann
  unsigned Leaf : 1;          // Leaf Flag
  unsigned : 5;               // pad to int8 width
  int8_t mark;                // Refine action flag: -1 Coarsen, 1 Ref, 0 No act
  int8_t normlsgn;            // Sign (+/-) multiplier for norml for init mesh
  uint8_t lvl;                // Level index
}
EDGE_t;
```

Figure 2.5: EDGE_t Structure

```
typedef struct edggeomdata {
  double edgelen;
  double norml[2];
  NODE_t *endpts[2];
}
EDGDATAG_t;
```

Figure 2.6: EDGEDATAG_t Structure



(a) K (b) $\{K_0, K_1, K_2, K_3\}$

Figure 2.7: Outward Normal Propagation

```
/* Triangle data structure */
typedef struct tristruct {
  int blk;                     // blk/tile indicator
  struct trigeomdata *data0;   // Struct containing area, atrf for init mesh
  GNode *treeloc;              // pointer to triangle location in KTree
  uint8_t lvl;                 // Level index TRIANGLE
  unsigned Leaf : 1;           // Leaf Flag
  unsigned recalc : 1;         // sd recalc flag
  unsigned nbrstate : 1;       // nbrstate flag
  unsigned cbdy : 1;           // cache bndry flag
  unsigned sblk : 4;           // cache subblock
  int8_t mark;                 // Refine action flag: -1 Coars, 1 Ref, 0 No act
  int8_t atrfsgn;              // Sign (+/-) multiplier for atrf for init mesh
  EDGE_t *edges[3];            // pointers to edges: 0-2
  NODE_t *corners[3];          // pointers to corners: 0-2
}
TRIANGLE_t;
```

Figure 2.8: TRIANGLE_t Structure

2.3.3 TRIANGLE_t Data Objects

The data structure for *TRIANGLE_t* objects is illustrated in Figure 2.8. Each instance of a TRIANGLE_t data object occupies 40 bytes of storage. The `data0` field provides a pointer to each triangle's ancestor in the initial mesh, the data struct corresponding to this object is described in Figure 2.9. Similar to the manner in which quantities such as edge length and outward normals were scaled from the ancestral edge in the initial mesh, triangle quantities such as the area and affine transformation coefficients can also be scaled. Specifically, the area of a triangle on level ℓ is $1/4^\ell$ times the area of the original ancestral triangle on the initial mesh. The scaling of the affine transformation coefficients will be described in section §2.7 and make use of the `atrfsgn` field.

There are fields which have particular usage in our implementation. The field `treeloc` is a pointer to a `GNode` object in the mesh hierarchy 4-ary tree `KTree`. `GNode` objects are part of the `glib` package (see §2.11.1). This allows the hierarchy relations to be stored separate from the `recalc` field which allows for selective recalculation of element stiffness matrix components depending on if mesh changes require it to be done. The `nbrstate` field indicates which instance of the element stiffness matrix diagonal block is active and is used primarily with the multilevel solvers (see §2.9). The `blk`, `cbdy`, and `sblk` fields are used when the mesh is partitioned for optimized usage of cache (see §2.10).

2.3.4 PDE Data

PDE data consists primarily of one or more vectors associated with each element $K \in \mathbb{T}$. There are three types of data objects relating to PDE data. The first data object is the diagonal block symmetric positive stiffness matrix associated with each element. Only the lower triangular portion plus the diagonal is stored. The diagonal block describes the interactions of an elements degrees of freedom (dof). The second data object is the off-diagonal block associated with each interior edge. The off-diagonal is an $n \times n$ matrix where n is the number of dof for the elements and describes interactions between dof of K^+ and dof of K^-. See §3.1.4, §4.1.3 for more information on stiffness

```
typedef struct trigeomdata {
  double area;
  double atrf[2][2];
}
TRIDATAG_t;
```

Figure 2.9: TRIDATAG_t Structure

matrix blocks and stiffness matrix assembly.

The third type of data object is simply one or more vectors of length n (dof) associated with each element. Of primary importance is maintaining the solution obtained during the solve process and element right hand side (rhs) vectors. More detailed information on the organization and use of these vectors is contained in §2.4, §2.5, and §2.9.

2.3.5 Object Relations

The interrelationship of the basic data objects is illustrated in Figure 2.10. The *offset* label on pointers into the PDE data vectors is described more fully in §2.5, however it should be pointed out that there is a one-to-one correspondence between objects and their associated PDE data vectors defined by these relations.

2.4 Memory Management

2.4.1 Background

During the early development stages of `Ell2` it became clear that a comprehensive and flexible memory management strategy was required. Initial efforts utilized a complete dynamically allocated memory strategy, i.e., when memory was required, it was requested through standard `C` function calls such as `malloc` and `calloc`. [5] There were problems though with adaptive code performance due to the unpredictability as to when and where the program would make system calls requesting memory. In addition, memory came back in blocks which were not contiguous and were difficult to manage and the code was susceptible to bugs which were hard to track down.

The logical next step was to allocate large blocks or chunks of memory for specific purposes at the beginning of the program. Being an old `FORTRAN` programmer, I was familiar with these techniques and this was not difficult to implement. Seeing as most computer programs are run in an iterative fashion until one is prepared to go into a production run phase, it did not seem to be an unreasonable burden to require as input limits on the total number of data objects that one would require, i.e., total number of vertices, internal and boundary edges, and triangles. Note that one of the added benefits is that the memory allocated is contiguous in virtual memory space, the operating system kernel determines whether physical contiguity is maintained.

The final step was to determine how to manage the memory which was allocated. With memory being allocated for a wide variety of purposes from data objects to linked lists and tree structures to temporary calculational storage, not all of which would be used at any one particular time, it made sense to separate the different memory requirements into different categories. The solution to this

[5] For detailed information on memory allocation on UNIX systems, the reader is referred to Stevens (1993).

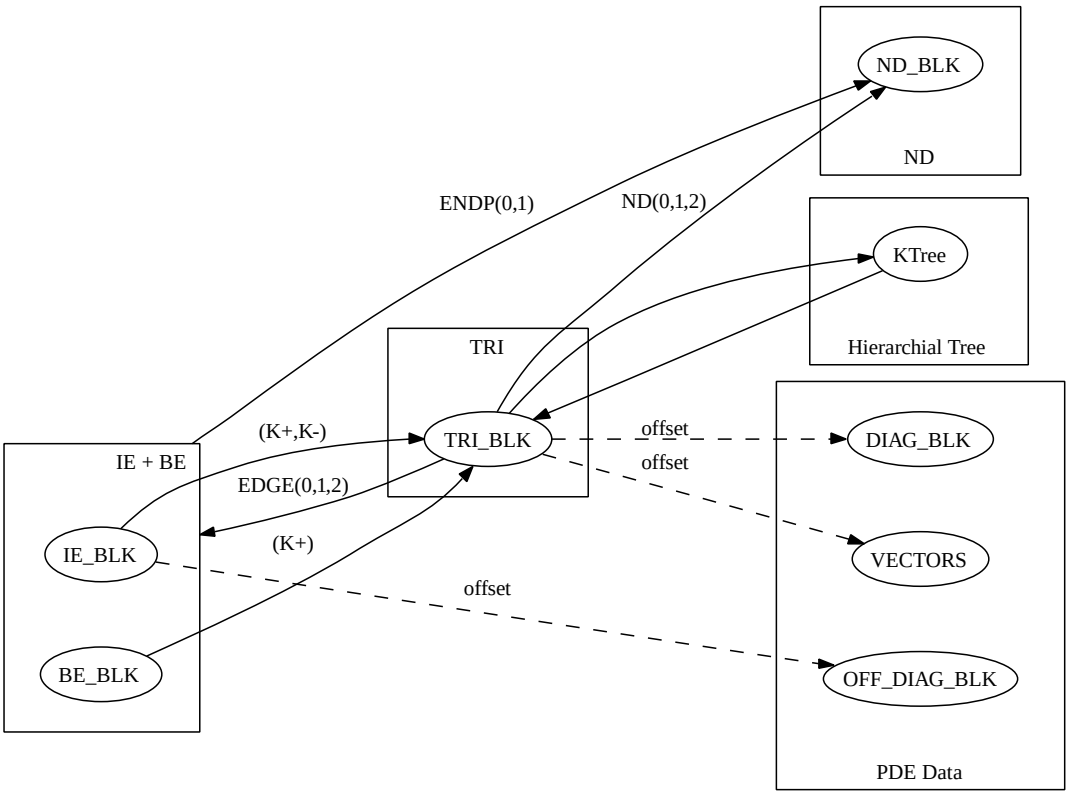


Figure 2.10: Data Object Relations

problem was the `Vmalloc` package developed by Kiem-Phong Vo from AT&T laboratories. Quoting Vo (1996):

> `Vmalloc` generalizes `malloc` to give programmers more control over memory allocation. `Vmalloc` introduces the idea of organizing memory into separate *regions*, each with a *discipline* to get raw memory and a *method* to manage allocation.

Note that for the purposes of this research, `Vmalloc` memory allocation is used primarily for organizing the required memory into separate regions. The extra options contained in the `Vmalloc` library to control memory access disciplines and methods to tailor memory allocation for specific data types and usage were not explored to any great degree. The interested reader is referred to Vo (1996) and `http://public.research.att.com/index.cfm?portal=19` for more information and download availability.

Essentially there are six areas of memory allocation utilized in our FEM implementation:

1. Global Variables - Static
2. Local Variables - Static
3. Template Storage - Dynamic
4. Object Storage - Dynamic
5. List Storage - Dynamic
6. Vector Storage - Dynamic

Managing and improving the allocation and management of the last three memory areas is where the most performance improvements can be gained.

There are three main types of dynamic memory allocation which are utilized in the programs developed in this research. The first type is explicitly allocated by the program using `Vmalloc` and assigned to particular regions, described in §2.4.2. The second type of memory is implicitly allocated and managed by the `glib` (§2.11.1) package and is used for small linked-list and n-ary tree usage. The last type is that which is allocated by included library routines, mainly the `triangle` (§2.11.2) and `METIS` (§2.11.7) packages. Note that the last two groups of dynamic memory allocations are also put under the `Vmalloc` paradigm.

On a final note, the memory allocation for any run is driven by the input parameters listed in Table 2.3. It should be noted that while allocation is performed for a large number of data objects at the beginning of a run, during the course of a run these data objects are grouped by the level associated with which each object. Thus there is a need for partitioning the global storage by level. After the initial triangulation $\{\mathcal{N}_0, \mathcal{E}_0^I, \mathcal{E}_0^B, \mathcal{T}_0\}$ is obtained by `triangle` (see §2.11.2), the following level based limits govern the *full refinement* level limits for each type of data object:

- `NODE_t`: $|\mathcal{N}_\ell|_{\max} = |\mathcal{E}_{\ell-1}^I|_{\max} + |\mathcal{E}_{\ell-1}^B|_{\max}$,
- Internal `EDGE_t`: $|\mathcal{E}_\ell^I|_{\max} = 3|\mathcal{T}_{\ell-1}|_{\max} + 2|\mathcal{E}_{\ell-1}^I|_{\max}$,
- Boundary `EDGE_t`: $|\mathcal{E}_\ell^B|_{\max} = 2|\mathcal{E}_{\ell-1}^B|_{\max}$,
- `TRIANGLE_t`: $|\mathcal{T}_\ell|_{\max} = 4|\mathcal{T}_{\ell-1}|_{\max}$.

Table 2.3: Memory Control Input Parameters

Input Parameter	Description
`LvlDepth`	Governs the maximum total depth of the Mesh Hierarchy tree $\mathbb{T}$. If this depth is reached and the adaptive input tolerance has not yet been reached, the program will stop execution.
`InitMemUnits[0]`	Determines how many total `NODE_t` objects are allocated.
`InitMemUnits[1]`	Determines how many total internal `EDGE_t` objects are allocated.
`InitMemUnits[2]`	Determines how many total boundary `EDGE_t` objects are allocated.
`InitMemUnits[3]`	Determines how many total `TRIANGLE_t` objects are allocated.
`InitLvlFull`	Determines how many initial levels of $\mathbb{T}$ will be fully refined based on the initial mesh.
`LvlMemUnits[0]`	Specifies a *level based* upper limit on `NODE_t` objects which are allocated for levels $\ell >$ `InitLvlFull`.
`LvlMemUnits[1]`	Specifies a *level based* upper limit on internal `EDGE_t` objects which are allocated for levels $\ell >$ `InitLvlFull`.
`LvlMemUnits[2]`	Specifies a *level based* upper limit on boundary `EDGE_t` objects which are allocated for levels $\ell >$ `InitLvlFull`.
`LvlMemUnits[3]`	Specifies a *level based* upper limit on `TRIANGLE_t` objects which are allocated for levels $\ell >$ `InitLvlFull`.

2.4.2 Memory Regions

Table 2.4 lists the main regions of memory which are allocated by the programs used in this research. There are other regions utilized which are not listed in the table, but they are not critical to the main implementations of adaptive DG FEM. Partitioning of memory allocation into regions using `Vmalloc` has proven to be very flexible in managing the diverse memory needs of an adaptive FEM implementation. Included in the `Vmalloc` package are function calls which allow for debugging particular regions as well as summary statistics on each region allocated.

2.4.3 Control Blocks

There is one other partitioning of the memory that has proven to be useful, grouping of certain data according to functional usage. This grouping has two purposes, first of which is to provide a method of allocating on a global basis key parameters, rather than them passing as arguments to individual functions. The second purpose is to provide a management strategy of individual variables which are not dynamically allocated but are statically declared. The goal here is to provide somewhat of a `FORTRAN Common Block` functionality in a `C` program. Rather than calling this grouping a `Common Block`, we call them *control blocks*.

Table 2.5 lists the major control blocks employed in this research. Figure 2.11 illustrates a portion of the declaration of the *Data Control Block* or *DCB*. Referencing a variable present in the DCB from a routine other than where it is declared simply requires an `extern` declaration of the complete block. Note also the presence of macro `#define` for simplifying reference to these variables.

2.5 Data Structures

Working with large amounts of diverse data in an adaptive FEM environment requires careful planning on how the data is managed. As was already mentioned in §2.4, large chunks of memory are

Table 2.4: `Vmalloc` User Defined Memory Regions

Memory Region	Description
`Reg_LL`	Contains the *maximum* level based limits up to level `LvlDepth` for vertices, edges, and triangles.
`Reg_GDATA`	Contains the initial mesh geometric data for edges, and triangles, such as area and edge lengths When this data is needed in calculations involving data objects of level $\ell > 0$, the appropriate items in this region are scaled appropriately by level.
`Reg_GOBJ`	Contains the *global* memory chunks for vertices, edges, and triangles.
`Reg_LNKB`	Contains the *global* memory chunks for linked list objects required to maintain level based lists for vertices, edges, and triangles.
`Reg_CNT`	Contains the level based *counts* for the current state of the linked lists.
`Reg_VEC`	Contains the *global* memory chunks associated with vectors involved in calculations such as solution and residual vectors.
`Reg_OFF`	Contains the *global* memory chunks associated with off-diagonal matrix blocks associated with interior edges.
`Reg_SD`	Contains the *global* memory chunks associated with diagonal matrix blocks associated with each triangle.
`Reg_TMPL`	Contains quadrature templates and penalty term templates on the reference element $\hat{K}$.
`Reg_Misc`	Contains miscellaneous allocations not specifically grouped in another region. Primarily this region contains `glib` allocations for temporary lists.

Table 2.5: Control Block Summary

Control Block	Description
JCB	Job Control Block - variables which are applicable to the total run
ECB	Estimator Control Block - variables which are related to a posteriori error estimation
MCB	Mesh Control Block - variables which are related to mesh maintenance
SCB	Solver Control Block - variables which are related to the linear solvers
MLCB	Memory/List Control Block - variables which are related to memory and list management
DCB	Data Control Block - variables which are related to PDE data
PCB	PAPI Control Block - variables which are related to performance monitoring with `PAPI`
TCB	Triangle Control Block - variables which are related to the use of `triangle` for the initial mesh
siloCB	Silo Control Block - variables which are related to the use of `silo/meshtv` for visualization

```
struct _datactrlblk {
  TRIDATAG_t *TR_GEOM_DATA;    // Init Mesh area,atrf Storage
  EDGDATAG_t *EDG_GEOM_DATA;   // Init Mesh edgelen, norml Storage
  double *vectmem;             // ptr to vector memory chnk
  double *vect[9];             // Large vectors
  double *OFF_DATA;            // Off diagonal edge interaction matrices
  double *SD_DATA;             // SD Matrices (symmetric storage)
};

/* DCB is declared in memlst.h */
#define TR_GEOM_DATA DCB.TR_GEOM_DATA
#define EDG_GEOM_DATA DCB.EDG_GEOM_DATA
#define vectmem DCB.vectmem
#define up DCB.vect[0]
#define bp DCB.vect[1]
#define rp DCB.vect[2]
#define sp DCB.vect[3]
#define pp DCB.vect[4]
#define qp DCB.vect[5]
#define zp DCB.vect[6]
#define vp DCB.vect[7]
#define estp DCB.vect[8]
#define OFF_DATA DCB.OFF_DATA
#define SD_DATA DCB.SD_DATA
```

Figure 2.11: DCB Control Block

allocated for each type of data object upon program startup. The problem now being faced is how to organize these data objects into structures which will allow efficient manipulation and computation. Of primary importance is the ability to change efficiently the grouping together of collections of data objects and the ability to iterate through these collections.

To achieve these goals, we have incorporated two fundamental data structures, linked lists and arrays. Linked lists commonly maintain pointers to the next element in a list, arrays utilize an index to access a specific item in the array. Linked lists are extremely efficient when accessing items in a sequential manner, but suffer from the drawback of having to traverse the list from the beginning (or end) to access an item located in the middle of the list. Arrays are the standard way of organizing and accessing data in a computer program, however when multiple dimensioned arrays become involved then poor memory utilization and index bookkeeping make the programmers task more difficult. In some respects however, one may consider an index to an element in an array as a pointer to that element because the compiler will translate the index into an address offset from the start of the array.

One commonly overused concept in `C` code is the use of pointers to pointers. While this provides great flexibility (and readability) in the code to identify relations between data objects, it often can produce poor program performance due to the jumping around in memory following the pointers. Thus, while one could conceivably construct a set of data structures which have a minimal set of pointers defining the relations between the data objects, our design of data structures has some redundancy built in. For example, since a triangle object has pointers to its three vertex `NODE_t` objects, and the triangle knows its three edges, there really is no need to maintain pointers for each edge to its endpoint `NODE_t` objects. However, we do maintain those pointers because they are useful in modifying the geometric mesh. The point here is that one must make a judgment as to when maintaining a pointer is necessary. If a one-to-one relation exists between data objects, then indexing might be adequate. If a many-to-one or one-to-many relation exists between data objects, then the use of pointers is probably the better choice.

To tie these thoughts together a bit, the reader is referred again to Figure 2.10, which describes the relations (address pointers) between the different data objects. Note that there are dashed lines between some of the data objects with the word *offset* above them. For example, associated with a `TRIANGLE_t` object has associated with it a diagonal block matrix. Looking back at the `TRIANGLE_t` data structure (see Figure 2.8) one should notice that there is no explicit pointer to the diagonal block matrix. The dashed line implies that this relation is implicit and does not need to be kept as a pointer, but rather is determined by the position of the data object relative to the start of the large chunk of memory allocated to those data objects, i.e., its offset.

This leads to the most important rule for understanding how one can improve performance when dealing with a set of data objects and associated sets of data objects:

Rule. All *auxiliary* sets of data objects associated with a set of *primary* data objects (`NODE_t`, `EDGE_t`, or `TRIANGLE_t`) maintain a one-to-one relation between elements of the primary set and elements of the auxiliary set. Thus working with a primary data object and requiring access to its associated auxiliary data object does not require maintenance of an address pointer and can be accessed directly by the primary objects offset relative to the beginning of the primary set in memory.

In other words, when large chunks of memory are allocated for the primary data objects, large chunks of memory are also allocated for the associated data objects which satisfy this rule. Table 2.6 describes the associated auxiliary data structures required for each set of primary data objects. Figure 2.12 illustrates the relation between the set of primary `NODE_t` objects in `NODE_BLOCK` and the auxiliary set of `GList`[6] data objects in `ND_LNK_BLK`.

[6] A `GList` object is part of the `glib` package and is shown in Figure 2.41.

Table 2.6: Large Memory Chunk Relations

Primary	Type	Aux	Type	Purpose
NODE_BLOCK	NODE_t	ND_LNK_BLK	GList	USED/AVAIL Doubly Linked List Objects
IEDGE_BLOCK	EDGE_t	IE_LNK_BLK	GList	USED/AVAIL Doubly Linked List Objects
		IE_LEAF_LNK_BLK	GList	LEAF/NLEAF Doubly Linked List Objects
		OFF_DATA	double *	Off-Diagonal Block Matrices
BEDGE_BLOCK	EDGE_t	BE_LNK_BLK	GList	USED/AVAIL Doubly Linked List Objects
		BE_LEAF_LNK_BLK	GList	LEAF/NLEAF Doubly Linked List Objects
TRI_BLOCK	TRIANGLE_t	TR_LNK_BLK	GList	USED/AVAIL Doubly Linked List Objects
		TR_LEAF_LNK_BLK	GList	LEAF/NLEAF Doubly Linked List Objects
		SD_DATA	double *	Diagonal Block Matrices
		vectmem	double *	Vectors

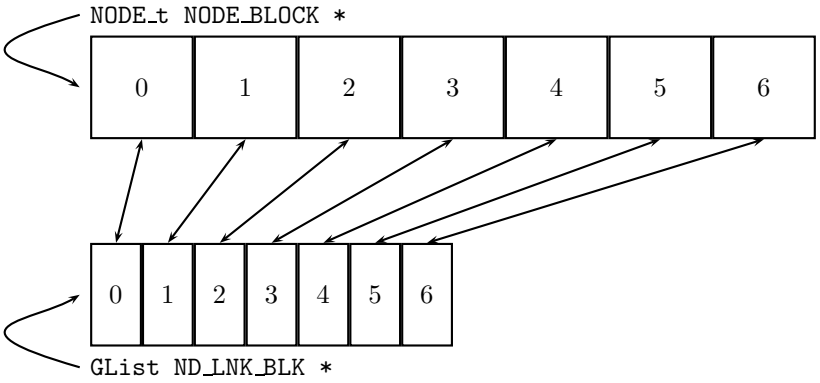


Figure 2.12: NODE_BLOCK – ND_LNK_BLK Relationship

2.5.1 List Structures

Associated with each set of *Link* objects listed in Table 2.6 are various sets of *List pointers* of type `GList *`, organized by *level*. These level list pointers determine sets of primary data objects associated with a particular level in the mesh hierarchy, with the initial mesh occupying level zero, and follow the concepts described in §2.2. In addition, all lists pointers for levels greater than zero are initialized to `NULL`. Table 2.7 describes the level based lists and associated pointers used in the programs developed for this research. Note that when I refer to a list, I am really referring to a pointer which points the start of a doubly linked list data object. As before, the input parameter $L_{\max}$ is defined to be the maximum hierarchy level that the program will run to. Note also, since the linked list objects maintain a one-to-one correspondence with the data objects, there is no need for maintaining a pointer to the data objects themselves [7].

The basic manner in which assignment of data objects to particular locations in memory is fairly simple. A *pool* of primary data objects is initially allocated (the large memory chunk) and all marked as AVAIL. For the initial mesh, items are *pulled* from the AVAIL pool and considered to be USED. The AVAIL list is then updated to point to the next available item in the pool. This sequence continues for all initial mesh primary data objects until complete.

Let L be the current maximum level in the mesh hierarchy. For subsequent levels, whenever the *first* data object is requested for level $L+1$ the upper *level boundary* for level L is set, and objects for level $L+1$ are then pulled starting from the level boundary just set. One of the hard parts of this method is estimating how many extra AVAIL objects should be allocated to level L when the start of level $L+1$ is set because there is no a priori knowledge of how many objects on level L will eventually be required. This is further discussed in §2.5.2. Figure 2.13 illustrates the basic organization of level based lists by showing the state of the lists for triangles of a mesh hierarchy of level L, $L > 0$. Note that in this figure USED and AVAIL are assumed to be contiguous on each level, this assumption will be relaxed later in the discussion. Note also that the LEAF/NLEAF link objects corresponding to AVAIL objects are not used, this follows from the fact that the LEAF/NLEAF list on a particular level is a partition of the USED objects on that level.

Before we get into more of the details, it is important to note that for each physical set of link blocks, two lists are actually present. This is because the USED/AVAIL and LEAF/NLEAF lists are individually mutually exclusive, i.e., if an object is USED it is no longer AVAIL. Similarly, if an object is a LEAF object, it is not a NLEAF object. This allows efficient utilization of the list link block storage essentially multiplexing the two lists together.

2.5.2 Maintenance

Now that we have a framework with which we can store individual data objects and group collections of data objects in an efficient manner, we are now faced with how to deal with additions and removals from the various lists. For example, consider the case of refining a triangle by regular refinement. A triangle can only be refined if exists on the leaf of the mesh hierarchy tree, therefore it will have to be removed from the the LEAF list for that level and inserted into the NLEAF list for that level. In addition, the four children triangles will have to be removed from the AVAIL list on the next level and inserted into the USED list for the next level, as well as being inserted into the LEAF list for the next level. Figure 2.14 illustrates a typical situation on how the link blocks are distributed on a particular level. The gray links are USED, others are AVAIL. Note that having AVAIL blocks in between USED blocks only occurs when coarsening is active. They are kept here in order to provide the most general picture.

[7] `GList` link objects do allow for a pointer to be stored as data.

Table 2.7: Level Based List Pointers, ℓ, $\quad \ell = 0, \ldots, L_{\max}$

List Pointer	Purpose
ND_LIST[ℓ]	Start of USED list of NODE_t data objects by level
ND_LIST_AV[ℓ]	Start of AVAIL list of NODE_t data objects by level
IE_LIST[ℓ]	Start of USED list of internal EDGE_t data objects by level
IE_LIST_AV[ℓ]	Start of AVAIL list of internal EDGE_t data objects by level
IE_LIST_LAST[ℓ]	Pointer to last USED internal EDGE_t data object by level
IE_LEAF[ℓ]	Start of LEAF internal EDGE_t data objects by level
IE_NLEAF[ℓ]	Start of NLEAF internal EDGE_t data objects by level
BE_LIST[ℓ]	Start of USED list of boundary EDGE_t data objects by level
BE_LIST_AV[ℓ]	Start of AVAIL list of boundary EDGE_t data objects by level
BE_LIST_LAST[ℓ]	Pointer to last USED boundary EDGE_t data object by level
BE_LEAF[ℓ]	Start of LEAF boundary EDGE_t data objects by level
BE_NLEAF[ℓ]	Start of NLEAF boundary EDGE_t data objects by level
TR_LIST[ℓ]	Start of USED list of TRIANGLE_t data objects by level
TR_LIST_AV[ℓ]	Start of AVAIL list of TRIANGLE_t data objects by level
TR_LIST_LAST[ℓ]	Pointer to last USED TRIANGLE_t data object by level
TR_LEAF[ℓ]	Start of LEAF TRIANGLE_t data objects by level
TR_NLEAF[ℓ]	Start of NLEAF TRIANGLE_t data objects by level

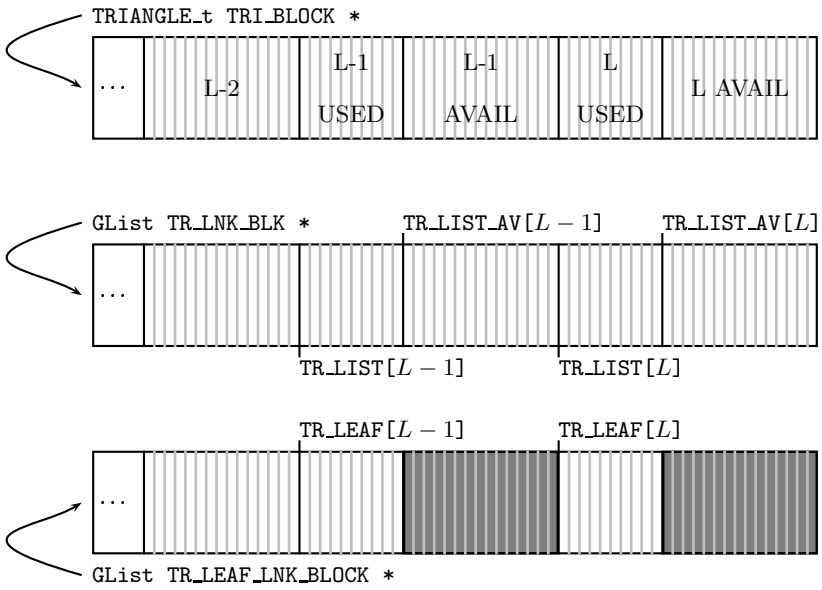


Figure 2.13: Triangle List State for level L mesh

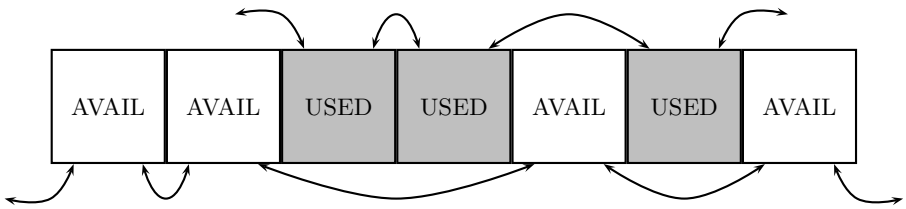


Figure 2.14: Link Blocks

Table 2.8: List Link Management Routines

Routine	Purpose
`Obj_Pull`	AVAIL Object → USED Object, adjust list and link pointers
`Obj_Free`	USED Object → AVAIL Object, adjust list and link pointers
`Swap_Pull`	Swaps AVAIL and USED Objects, called from `Obj_Pull`
`Add_to_Leaf`	Adds Object to LEAF list
`Swap_Free`	Swaps USED and AVAIL Objects, called from `Obj_Free`
`Rem_fr_Leaf`	Removes Object from LEAF list
`Leaf_to_NLeaf`	Moves Object from LEAF list to NLEAF list
`NLeaf_to_Leaf`	Moves Object from NLEAF list to LEAF list

In order to handle the various situations of moving data objects say from AVAIL to USED or LEAF to NLEAF, the routines in Table 2.8 were developed and implemented. Note that no actual movement of the data objects themselves, either the primary data objects or the link objects, is performed. All that takes place during these operations is adjustment of link and list pointers.

The maintenance of the lists of data objects, both USED/AVAIL and the LEAF/NLEAF provides for great flexibility in the adaptive FEM environment. By taking care of the allocation upon program initialization, needless calls to the operating system for dynamic memory is eliminated. In addition, list maintenance is fast, because all that is updated is the list and link pointers. Finally, some sense of data contiguity is maintained because in the pure refinement adaptive runs, all of the USED objects are located at the beginning of the list and all of the AVAIL objects are at the end of the list for each level. However, there may be contiguity *gaps* between LEAF and NLEAF objects.

2.5.3 Reordering

To obtain better data contiguity of the data objects within a list, we have implemented an ordering option for the LEAF and NLEAF objects on each level. This operation requires two phases, the first is mandatory and the second is optional. The first phase requires *sweeping* all of the NLEAF data objects to the end of the LEAF list. This phase does require physical movement of memory (using `memcpy`) to move data objects data from one area to another. However, the one-to-one correspondence is maintained in its entirety. The second phase requires *reordering* within the LEAF and NLEAF lists based on some external ordering criteria. This may be important in some cases if some optimal ordering[8] is required during the solve process for performance optimization (see §2.10.2).

Currently only LEAF triangle data objects are reordered. In order to reorder triangles, address

[8]Such as would be required if a *space filling curve* method is used.

pointers pointing involving edges must also be updated. Specifically, when a triangle is moved, all internal edges which are a part of the triangle's boundary must be updated to reflect the triangle's new location in memory, i.e, K^+ and K^- pointers. In addition, the diagonal block matrix associated with the triangle requires movement, as well as solution and right hand side vectors for the triangle. Finally, the hierarchy mesh tree must also be updated. This is accomplished via a *swapping* process which basically just a single sweep through each level's LEAF list.

One idea which has not been implemented but which might provide additional performance gains is the reordering of internal edges and associated off-diagonal block matrices. Once an ordering scheme is identified and developed for this, it would certainly be worth an effort to obtain better data contiguity of the off-diagonal block matrices.

2.5.4 Tree Structure and Auxiliary Lists

Currently we rely on `glib` to allocate and manage the hierarchy tree using `GNode` data objects. Certainly one improvement would be to perform the allocation upon program initialization, as is done for the majority of lists. However, `glib` provides for allocation in a buffered manner and it does a pretty good job. Finally, there are other miscellaneous lists used throughout the programs developed for this research, mainly singly linked lists and some additional doubly linked lists. These temporary lists reuse their data objects during repeated adaptive iterations, and thus it allowing `glib` to handle the link allocation and management allowed me to focus on more important issues. For more information on `glib` and its functionality, see §2.11.1.

2.6 Adaptive Methods

Adaptive methods in numerical approximation of partial differential equations basically follow the same strategy, that of *Solve – Estimate – Refine* or *SER*. Between the Estimate and the Refine steps is one that is very important, the *Marking* step[9]. The Marking step determines which elements should be refined (or coarsened).

One approach to implementing an adaptive process is to utilize a posteriori error estimators [10] to guide the selection of a subset of elements $\mathcal{S} = \{K\} \subset \mathcal{T}_h$ such that when all elements of $\mathcal{S}$ are refined, the resulting error will be approximately equidistributed among all elements. This process is repeated until some overall global tolerance is achieved. Here we present a marking strategy based on that proposed in Dörfler (1996). The reader who is interested in other marking strategies is referred to Babuşka and Rheinboldt (1978) and Morin et al. (2000).

2.6.1 Marking Strategies

We present first the marking strategy described in Dörfler (1996), reproduced in Algorithm 2. Note that this algorithm is guaranteed to stop because τ goes to zero. Choosing ν determines how *fine* the procedure will work, smaller values of ν allow the marking strategy to step through the range of the estimator with finer step size. The choice of θ determines the *fraction* of the global estimator that one wants to refine. Choosing θ close to one would produce uniform refinement, i.e., all $K \in \mathcal{T}_h$ are refined. Choosing θ small would only choose a few elements to be marked each adaptive iteration, thus resulting in many adaptive iterations (and many solves) to reach the input desired tolerance ϵ.

[9] Often the Marking steps is considered to be part of the Refine step.

[10] See §3.2, §4.2 for a more detailed description of different types of a posteriori error estimators.

Algorithm 2 Dörfler Marking Strategy (Dörfler, 1996)

Require: Fix $\theta \in (0,1)$
Require: Fix $\nu \in (0,1)$, small
  $\mathcal{S} = \emptyset$
  $s = 0$
  $\tau = 1$
  while $s < \theta^2 \eta_{\mathcal{T}}^2$ **do**
    $\tau = \tau - \nu$
    for all $K \in \mathcal{T}_h$ **do**
      if K is not marked **then**
        if $\eta_K > \tau \eta_{\max}$ **then**
          Mark K, $\mathcal{S} = \mathcal{S} + K$
          $s = s + \eta_K^2$
        end if
      end if
    end for
  end while

Note that when θ is small one will usually obtain a more optimal mesh, but at a very large cost in adaptive iterations and thus solve time. The complete adaptive process stops when $\eta_{\mathcal{T}} \leq \epsilon$ or $\mathcal{S} = \emptyset$.

Assume that one has computed a set of error estimators $\{\eta_K\}$ for each element $K \in \mathcal{T}_h$. Assume also that on this set one has calculated a global error estimator

$$\eta_{\mathcal{T}}^2 = \sum_{K \in \mathcal{T}_h} \eta_K^2 \tag{2.2}$$

and that one has determined the largest value

$$\eta_{\max} := \max_{K \in \mathcal{T}_h} \{\eta_K\}.$$

The goal here is to form the set $\mathcal{S} \subset \mathcal{T}_h$.

We now present some modifications to the Dörfler marking strategy which has been implemented in our code. First, let $\{\eta_K\}$ be sorted in descending order, call the element index set associated with this ordering of the η_K's $\widehat{\mathcal{T}_h}$. Note that by sorting the estimators in decreasing order and traversing the list in the same manner, one essentially duplicates the Dörfler strategy in that the largest estimator contributions are taken first. This can be seen in Figure 2.15 where the crosshatched area indicates the fraction θ of the global estimator total which is marked for refinement. In this initial strategy, θ is input and fixed. The process is illustrated in Algorithm 3.

There are a number of issues which need to be addressed with both marking strategies. It is clear that the choice of θ affects greatly the quality of the resulting mesh and ultimately the total number of adaptive iterations and solve time required to achieve the tolerance ϵ. The SER process continues, marking a fixed fraction of the global residual, driving the peak of the distribution illustrated in Figure 2.15 toward the left and smaller η_K. Note also that if the distribution of the estimator were truly equidistributed, the distribution would be a delta function located at the mean of the estimator distribution.

Another issue occurs when one is close to the desired tolerance, and the fixed value of θ produces $\eta_{\mathcal{T}} < \epsilon$ where $\eta_{\mathcal{T}}$ is much smaller than ϵ indicating that one refined more triangles than was necessary

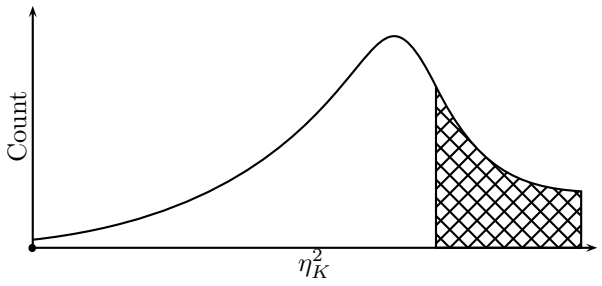


Figure 2.15: Typical Estimator Distribution and Marking

Algorithm 3 Fixed θ Marking Strategy

Require: Fix $\theta \in (0, 1)$
  $\mathcal{S} = \emptyset$
  $s = 0$
  if $\eta_{\mathcal{T}} \geq \epsilon$ **then**
    for all $K \in \widehat{\mathcal{T}_h}$ **do**
      Mark K, $\mathcal{S} = \mathcal{S} + K$
      $s = s + \eta_K^2$
      if $s \geq \theta^2 \eta_{\mathcal{T}}^2$ **then**
        `break`
      end if
    end for
  end if

to reach the input tolerance. While one may say that this is okay, input tolerance has been achieved, one would like to use ϵ as a control on the adaptive process and thus overshooting the adaptive tolerance by a great deal is not desirable.

It is clear that one requires some method of choosing a variable θ_V which would result in many triangles being refined initially in somewhat of a *greedy* manner, and which would result in a more selective refinement choice when one is *close* to the desired estimator tolerance. To provide motivation for our method, note the following a priori error estimate (Baker et al., 1990; Prudhomme et al., 2000):

$$\|u - u_h^\gamma\|_{1,h} \leq c \left(\sum_{K \in \mathcal{T}_h} h_K^{2(r-1)} |u|_{r,K}^2 \right)^{1/2} \tag{2.3}$$

where c is a constant independent of h and u. Thus, when refining uniformly with $h \to h' = h/2$, one expects the error in the energy norm to reduce by approximately a factor of 2^p, where $p = r - 1$. Thus, we utilize the following heuristics regarding the error estimator η:

$$\|\eta_{\mathcal{T}}\|_{1,h',\Omega} \approx 2^{-p} \|\eta_{\mathcal{T}}\|_{1,h,\Omega} \tag{2.4}$$

$$\|\eta_K\|_{1,h',K} \approx 2^{-p} \|\eta_K\|_{1,h,K}. \tag{2.5}$$

Note that the energy norms on the left hand side of these equations involves the higher dimensional space $V_{h'}^r$.

This leads us to define the following useful notation:

- Define $\zeta := \dfrac{\eta_{\mathcal{T}}}{\epsilon}$ as a test indicator as to how close one is to achieving the adaptive tolerance ϵ.
- Define $\xi := 2^p$, where $p = r - 1$ is the degree of the polynomials used on each triangle. ξ is the expected a priori global reduction in $\|\nabla e\|$ or $\|\eta\|_{1,h}$ due to uniform refinement ($h \to h' = h/2$) of the entire mesh.
- Define $\bar{q} := \dfrac{\epsilon^2}{|\mathcal{T}|}$ as the target tolerance in an idealized situation where the error is equidistributed among all triangles $K \in \mathcal{T}_h$.

When $\zeta > \xi$, uniform refinement of all triangles should not reduce the estimator by more than a factor of ξ, and thus one can be aggressive in choosing triangles to be marked for refinement, i.e., θ_V will be close to one. When $\zeta \leq \xi$, one has to be be more selective in choosing θ_V so as to try and hit the desired adaptive tolerance in few adaptive iterations (ideally, one adaptive iteration).

We can now fully describe our marking strategy, detailed in Algorithms 4 and 5. We will work with the distribution of $\log_2 \eta_K^2$ as shown in Figure 2.16 instead of the distribution of η_K^2 as shown in Figure 2.15. This allows us to determine a threshold distance of $\bar{q} + 2p$ beyond which triangles which are refined will in general not end up producing triangles with $\log_2 \eta_K^2 < \bar{q}$.

$\zeta > \xi$.

When in this range, one is interested in global reduction of the estimator and thus one refines any triangle greater than the threshold $\bar{q} + 2p$. This is illustrated in Figure 2.16, where all triangles in the crosshatched area will be refined. Note that the mean of the distribution will shift to the left.

$\zeta \leq \xi$.

When in this range, one is interested in local reductions of the estimator and thus one refines any triangle greater than the threshold $\bar{q}+2p$ such that the predicted reduction accumulated on a triangle

Algorithm 4 Variable θ Marking Strategy

Require: Adaptive Iteration i, $\eta_T = \eta_{T,i}$, $\eta_{T,i-1}$
  $\theta_V =$ `THETA_R`
  $\mathcal{S} = \emptyset$
  $s = 0$
  if $\eta_T/\epsilon \leq 1.01$ **then**
    `return`
  end if
  if $\eta_T \geq \epsilon$ **then**
    for all $K \in \widehat{\mathcal{T}_h}$ **do**
      Mark K, $\mathcal{S} = \mathcal{S} + K$
      $s = s + \eta_K^2$
      if $s \geq \theta_V^2 \eta_T^2$ **then**
        `break`
      end if
    end for
  end if

Algorithm 5 θ_V Determination: `THETA_R`

Require: $\zeta, \xi, \eta_T^2, \bar{q}, p$
  $thresh = \bar{q} + 2p$; $s_r = 0$
  if $\zeta > \xi$ **then**
    for all $K \in \widehat{\mathcal{T}_h}$ **do**
      $s_r = s_r + \eta_K^2$
      if $\log_2 \eta_K^2 < thresh$ **then**
        `break`
      end if
    end for
  else
    $p_e = \eta_T^2$
    for all $K \in \widehat{\mathcal{T}_h}$ **do**
      $s_r = s_r + \eta_K^2$
      $p_e = p_e - \eta_K^2 + \eta_K^2/\xi^2$
      if $(p_e)^{1/2}/\epsilon \leq 1$ **then**
        `break`
      end if
    end for
  end if
  $\theta_V = (s_r/\eta_T^2)^{1/2}$

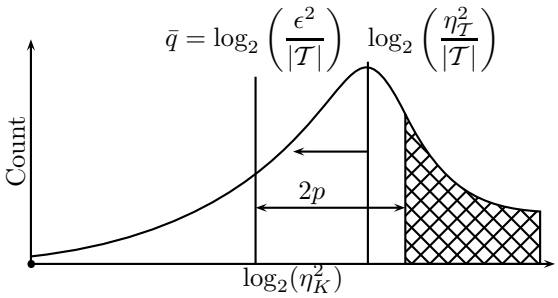


Figure 2.16: θ_V Marking: $\zeta > \xi$

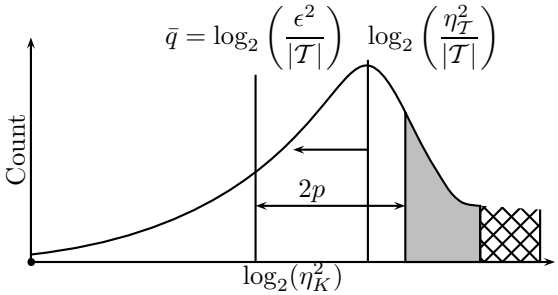


Figure 2.17: θ_V Marking: $\zeta \leq \xi$

by triangle basis will hit the input tolerance ϵ. This is illustrated in Figure 2.17, where all triangles in the crosshatched area will be refined. Note once again that the mean of the distribution again will shift to the left.

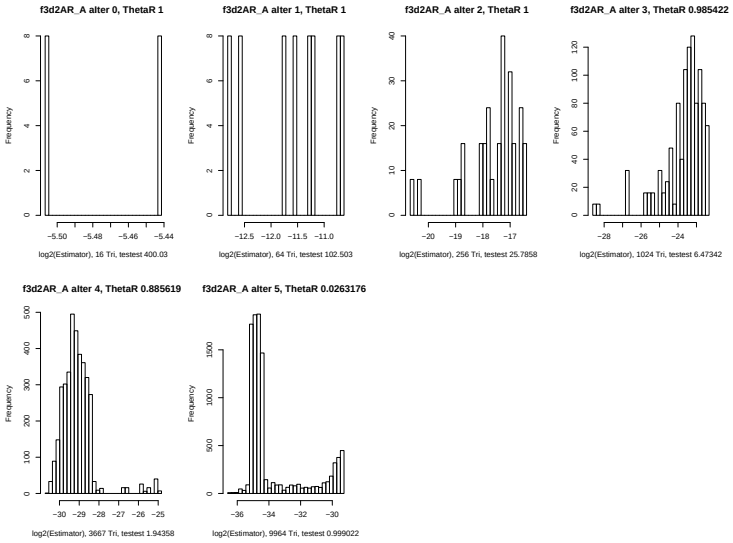
A couple of notes regarding this marking strategy. First, uniform refinement occurs on the first few levels until the global estimator comes within *range* of achieving the tolerance on one step based on a priori analysis. Since experience shows that when starting with a coarse mesh the first few levels are refined uniformly anyway, this algorithm is optimal when far away from the target tolerance. Finally, one should note that we require the final mesh to satisfy $\frac{\eta_{\mathcal{T}}}{\epsilon} \leq 1.01$.

Theoretical justification and rigorous proof of the effectiveness of this method for now lies only in the numerical results. Figures 2.18–2.19 provides some sample estimator $\log_2 \eta^2$ distributions utilizing variable θ_V marking and which correspond to the adaptive meshes displayed later in this thesis.

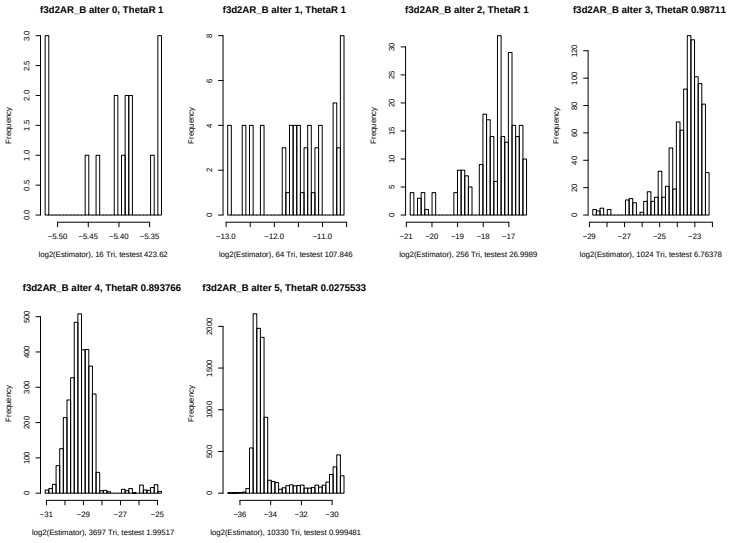
2.6.2 Refinement

As has been mentioned before, DG allows for regular refinement of triangles. The overall process by which this is accomplished is as follows:

1. Traverse $\mathcal{T}$ (current mesh) and mark $K \in \mathcal{T}$ for refinement/coarsening. The List $RTri$ then contains a list of triangles to be refined, the List $CTri$ contains a list of triangles to be coarsened.
2. Identify additional $K \in \mathcal{T}$ that need to be refined in order to satisfy the two neighbor condition.
3. Sort $RTri$ by increasing level, then by lexigraphic order.
4. Refine each $K \in RTri$ producing four children K_0, K_1, K_2, K_3, updating T trees and T, E^I, E^B, N lists.

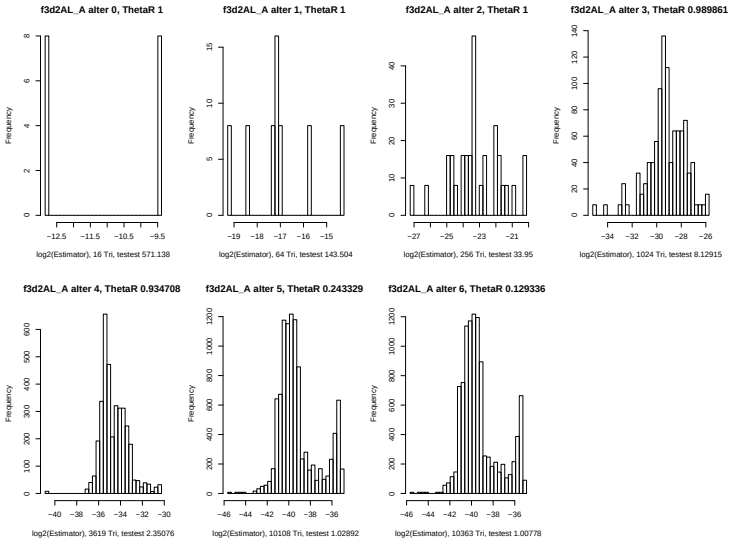


(a) f3, r=3, Arnold

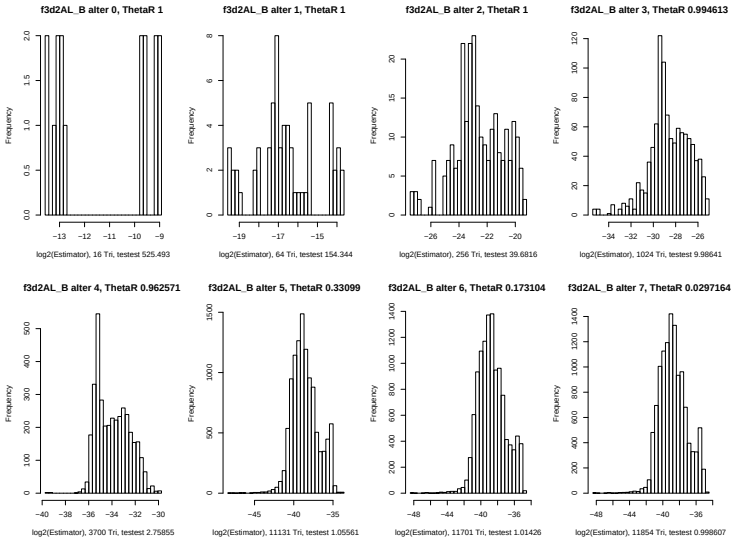


(b) f3, r=3, Baker

Figure 2.18: Variable θ_V η^2 Histogram, Residual



(a) f3, r=3, Arnold



(b) f3, r=3, Baker

Figure 2.19: Variable θ_V η^2 Histogram, Local

The detail of the refinement process involves management of the geometric mesh objects, USED/AVAIL, and LEAF/NLEAF and does not require much explanation here other than to state that existing data objects are reused when they can be, i.e., an edge instance exists only once in the data object storage, even though more than one element may share the edge.

2.6.3 Coarsening

We impose a condition on coarsening such that coarsening can occur only if all four children are marked for coarsening. In that case, the four children elements are replaced by their common parent element in the mesh.

We have not implemented coarsening actions for elliptic problems considered in this dissertation, although the algorithms have been developed and coded and tested on a moving mesh program called `rotsing` or rotating singularity. This program allowed us to reuse data objects in the refinement process which were previously released during a coarsening process. Basically, we simulated a point singularity in Ω and described a refinement ball around the singularity. We then moved the singularity and its neighborhood around the origin and handled the refinement and coarsening actions accordingly. When the singularity returned to where it was started, the exact same mesh was obtained with a nice bound on the memory and data object usage. This provides proof of concept which will be applied later when research moves to time dependent problems.

The coarsening impact on object management is the inverse of the refinement process, and thus will not be talked about here.

2.7 Affine Transformations

In finite element methods, quadrature over each triangle and edge may require approximating the integrals on very small triangles which can result in quadrature errors. A common technique in FEM is to perform quadrature over a *reference element* and utilize *affine transformations* to transform the quadrature rule approximations between the reference element and the actual triangles in the mesh. This technique also allows for calculation of *templates* for some integrals on the reference element which can be reused during the calculational process. Here we describe the affine transformations used in this research and implemented into our codes.

Let $\hat{X}$ be a polynomial space over $\hat{K}$, dim $\hat{X} = I$. Choose a set of nodal points $\{\hat{\mathbf{x}}_i\}_{i=1}^I$ in $\hat{K}$ such that any function $\hat{v} \in \hat{X}$ is uniquely determined by its values at the nodes $\{\hat{\mathbf{x}}_i\}_{i=1}^I$, and

$$\hat{v}(\hat{\mathbf{x}}) = \sum_{i=1}^{I} \hat{v}(\hat{\mathbf{x}}_i)\hat{\phi}_i(\hat{\mathbf{x}}).$$

The functions $\{\hat{\phi}_i\}_{i=1}^I$ form a basis for the space $\hat{X}$ with the property

$$\hat{\phi}_i(\hat{\mathbf{x}}_j) = \delta_{ij}.$$

Now consider the invertible affine mapping $F_K : \hat{K} \to K$ of the form

$$F_K(\hat{\mathbf{x}}) = \mathbf{T}_K\hat{\mathbf{x}} + \mathbf{b}_K.$$

The mapping F_K is a bijection between $\hat{K}$ and K, $\mathbf{T}_K$ is an invertible 2×2 matrix, and $\mathbf{b}_K$ is a translation vector.

Over each element K, one can define a finite-dimensional function space X_K by

$$X_K = \hat{X} \circ F_K^{-1} \equiv \{v | v = \hat{v} \circ F_K^{-1}, \hat{v} \in \hat{X}\}.$$

Thus,

$$v(\mathbf{x}) = \hat{v}(\hat{\mathbf{x}}) \quad \forall \mathbf{x} \in K, \hat{\mathbf{x}} \in \hat{K}, \text{ with } \mathbf{x} = F_K(\hat{\mathbf{x}}).$$

Now, define

$$\mathbf{x}_i^K = F_K(\hat{\mathbf{x}}_i), \quad i = 1, \ldots, I$$

and

$$\phi_i^K = \hat{\phi}_i \circ F_K^{-1}, \quad i = 1, \ldots, I.$$

Then, the functions $\{\phi_i^K\}$ have the property that

$$\phi_i^K(\mathbf{x}_j^K) = \delta_{ij}.$$

2.7.1 Affine Transformations: F_K,F_K^{-1}

Consider the affine transformation mapping the reference triangle $\hat{K}$ onto any $K \in \mathcal{T}_h$ as illustrated in Figure 2.20. We define $F_K : \hat{X} \to X$ as

$$F_K(\hat{\mathbf{x}}) = A_K \hat{\mathbf{x}} + \mathbf{b}_K = \mathbf{x}$$

where

$$A_K = \begin{pmatrix} a_{11}^K & a_{12}^K \\ a_{21}^K & a_{22}^K \end{pmatrix} = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}$$

and

$$\mathbf{b}_K = \begin{pmatrix} b_1^K \\ b_2^K \end{pmatrix} = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}$$

We can now find an explicit representation of $F_K^{-1} : X \to \hat{X}$

$$F_K^{-1}(\mathbf{x}) = A_K^{-1}(\mathbf{x} - \mathbf{b}_K) = \hat{\mathbf{x}}$$

where

$$A_K^{-1} = \frac{1}{\det A_K} \begin{pmatrix} a_{22}^K & -a_{12}^K \\ -a_{21}^K & a_{11}^K \end{pmatrix} = \frac{1}{2|K|} \begin{pmatrix} y_2 - y_3 & x_3 - x_2 \\ y_3 - y_1 & x_1 - x_3 \end{pmatrix} = \frac{1}{2|K|} \begin{pmatrix} \hat{a}_{11}^K & \hat{a}_{12}^K \\ \hat{a}_{21}^K & \hat{a}_{22}^K \end{pmatrix} \equiv \frac{1}{2|K|} \hat{A}_K.$$

Note that we can write

$$F_K(\hat{\mathbf{x}}) = \begin{pmatrix} f^K(\hat{\mathbf{x}}) \\ g^K(\hat{\mathbf{x}}) \end{pmatrix}$$

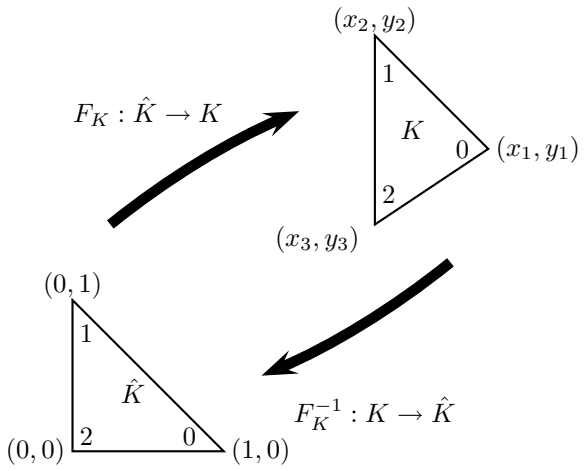


Figure 2.20: Affine Transformation from K onto Reference Triangle $\hat{K}$

and

$$F_K^{-1}(\mathbf{x}) = \begin{pmatrix} \hat{f}^K(\mathbf{x}) \\ \hat{g}^K(\mathbf{x}) \end{pmatrix}.$$

The component functions used above are thus:

$$\begin{aligned} f^K(\hat{\mathbf{x}}) &= a_{11}^K \hat{x} + a_{12}^K \hat{y} + b_1^K \\ g^K(\hat{\mathbf{x}}) &= a_{21}^K \hat{x} + a_{22}^K \hat{y} + b_2^K \\ \hat{f}^K(\mathbf{x}) &= \frac{1}{2|K|} \left(a_{22}^K (x - b_1^K) - a_{12}^K (y - b_2^K) \right) \\ \hat{g}^K(\mathbf{x}) &= \frac{1}{2|K|} \left(-a_{12}^K (x - b_1^K) + a_{22}^K (y - b_2^K) \right) \end{aligned}$$

The Jacobian matrices for F_K, F_K^{-1} are

$$J_F^K = \begin{pmatrix} \dfrac{\partial f^K}{\partial \hat{x}} & \dfrac{\partial f^K}{\partial \hat{y}} \\ \dfrac{\partial g^K}{\partial \hat{x}} & \dfrac{\partial g^K}{\partial \hat{y}} \end{pmatrix} = \begin{pmatrix} a_{11}^K & a_{12}^K \\ a_{21}^K & a_{22}^K \end{pmatrix} = A_K$$

and

$$J_{F^{-1}}^K = \begin{pmatrix} \dfrac{\partial \hat{f}^K}{\partial x} & \dfrac{\partial \hat{f}^K}{\partial y} \\ \dfrac{\partial \hat{g}^K}{\partial x} & \dfrac{\partial \hat{g}^K}{\partial y} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial \hat{x}}{\partial x} & \dfrac{\partial \hat{x}}{\partial y} \\ \dfrac{\partial \hat{y}}{\partial x} & \dfrac{\partial \hat{y}}{\partial x} \end{pmatrix} = \frac{1}{2|K|} \begin{pmatrix} a_{22}^K & -a_{12}^K \\ -a_{21}^K & a_{11}^K \end{pmatrix} = A_K^{-1} = \frac{1}{2|K|} \hat{A}_K.$$

Finally, note that $\hat{A}_K$ is the affine transformation coefficients stored with each triangle K.

2.7.2 Basis Functions

In finite element methods for Lagrangian basis elements with N degrees of freedom, the basis functions $\phi_j(\mathbf{x}) \in V_h, \quad j = 1, \ldots, N$

$$\phi_j(\mathbf{x}_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.6}$$

Assume that on K the following functions are defined

$$u(\mathbf{x}) = \sum_{j=1}^{N} u(\mathbf{x}_j^K)\phi_j^K(\mathbf{x})$$

and

$$v(\mathbf{x}) = \sum_{i=1}^{N} \phi_i^K(\mathbf{x})$$

where N is the number of degrees of freedom and for convenience we will drop the superscript K annotation unless otherwise needed.

2.7.3 Derivative Transformations

Since quadrature (1D and 2D) will be performed over part or all of the reference element $\hat{K}$, one will need expressions relating partial derivative on K to partial derivatives on $\hat{K}$.

First Order Partials

$$\begin{aligned} \frac{\partial \phi_j}{\partial x} &= \frac{\partial \hat{\phi}_j}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + \frac{\partial \hat{\phi}_j}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial x} \\ &= \frac{1}{2|K|}\left(\frac{\partial \hat{\phi}_j}{\partial \hat{x}}\hat{a}_{11} + \frac{\partial \hat{\phi}_j}{\partial \hat{y}}\hat{a}_{21}\right) \end{aligned}$$

and

$$\begin{aligned} \frac{\partial \phi_j}{\partial y} &= \frac{\partial \hat{\phi}_j}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial y} + \frac{\partial \hat{\phi}_j}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y} \\ &= \frac{1}{2|K|}\left(\frac{\partial \hat{\phi}_j}{\partial \hat{x}}\hat{a}_{12} + \frac{\partial \hat{\phi}_j}{\partial \hat{y}}\hat{a}_{22}\right). \end{aligned}$$

Finally then

$$\partial_n \phi_j = \frac{1}{2|K|}\left[\left(\frac{\partial \hat{\phi}_j}{\partial \hat{x}}\hat{a}_{11} + \frac{\partial \hat{\phi}_j}{\partial \hat{y}}\hat{a}_{21}\right)n_x + \left(\frac{\partial \hat{\phi}_j}{\partial \hat{x}}\hat{a}_{12} + \frac{\partial \hat{\phi}_j}{\partial \hat{y}}\hat{a}_{22}\right)n_y\right].$$

Second Order Partials

$$\begin{aligned}\frac{\partial^2\phi_j}{\partial x^2} &= \frac{\partial}{\partial x}\left(\frac{\partial\phi_j}{\partial x}\right) = \frac{\partial}{\partial\hat{x}}\left(\frac{\partial\phi_j}{\partial x}\right)\frac{\partial\hat{x}}{\partial x} + \frac{\partial}{\partial\hat{y}}\left(\frac{\partial\phi_j}{\partial x}\right)\frac{\partial\hat{y}}{\partial x} \\ &= \left(\frac{1}{2|K|}\right)^2\left[\hat{a}_{11}^2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2} + 2\hat{a}_{11}\hat{a}_{21}\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}} + \hat{a}_{21}^2\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right]\end{aligned}$$

and

$$\begin{aligned}\frac{\partial^2\phi_j}{\partial y^2} &= \frac{\partial}{\partial y}\left(\frac{\partial\phi_j}{\partial y}\right) = \frac{\partial}{\partial\hat{x}}\left(\frac{\partial\phi_j}{\partial y}\right)\frac{\partial\hat{x}}{\partial y} + \frac{\partial}{\partial\hat{y}}\left(\frac{\partial\phi_j}{\partial y}\right)\frac{\partial\hat{y}}{\partial y} \\ &= \left(\frac{1}{2|K|}\right)^2\left[\hat{a}_{12}^2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2} + 2\hat{a}_{12}\hat{a}_{22}\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}} + \hat{a}_{22}^2\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right]\end{aligned}$$

and the Laplacian

$$\begin{aligned}\Delta\phi_j &= \frac{\partial^2\phi_j}{\partial x^2} + \frac{\partial^2\phi_j}{\partial y^2} \\ &= \left(\frac{1}{2|K|}\right)^2\left[\left(\hat{a}_{11}^2 + \hat{a}_{12}^2\right)\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2} + 2\left(\hat{a}_{11}\hat{a}_{21} + \hat{a}_{12}\hat{a}_{22}\right)\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}} + \left(\hat{a}_{21}^2 + \hat{a}_{22}^2\right)\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right] \\ &= \left(\frac{1}{2|K|}\right)^2\left[c_1\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2} + c_2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}} + c_3\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right].\end{aligned}$$

Third Order Partials

$$\frac{\partial(\Delta\phi_j)}{\partial n} = \nabla(\Delta\phi_j)\cdot n = \left(\frac{\partial}{\partial x}\Delta\phi_j\right)n_x + \left(\frac{\partial}{\partial y}\Delta\phi_j\right)n_y$$

Now

$$\frac{\partial}{\partial x}\Delta\phi_j = \frac{\partial}{\partial\hat{x}}(\Delta\phi_j)\frac{\partial\hat{x}}{\partial x} + \frac{\partial}{\partial\hat{y}}(\Delta\phi_j)\frac{\partial\hat{y}}{\partial x}$$

and

$$\frac{\partial}{\partial y}\Delta\phi_j = \frac{\partial}{\partial\hat{x}}(\Delta\phi_j)\frac{\partial\hat{x}}{\partial y} + \frac{\partial}{\partial\hat{y}}(\Delta\phi_j)\frac{\partial\hat{y}}{\partial y}$$

implies after some calculations

$$\begin{aligned}\frac{\partial}{\partial x}\Delta\phi_j &= \left(\frac{1}{2|K|}\right)^3\left[\hat{a}_{11}c_1\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^3} + (\hat{a}_{11}c_2 + \hat{a}_{21}c_1)\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^2\partial\hat{y}} + (\hat{a}_{11}c_3 + \hat{a}_{21}c_2)\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}^2} + \hat{a}_{21}c_3\frac{\partial^3\hat{\phi}_j}{\partial\hat{y}^3}\right] \\ &= \left(\frac{1}{2|K|}\right)^3\left[d_{11}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^3} + d_{12}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^2\partial\hat{y}} + d_{13}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}^2} + d_{14}\frac{\partial^3\hat{\phi}_j}{\partial\hat{y}^3}\right]\end{aligned}$$

and

$$\frac{\partial}{\partial y}\Delta\phi_j = \left(\frac{1}{2|K|}\right)^3 \left[\hat{a}_{12}c_1\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^3} + (\hat{a}_{12}c_2 + \hat{a}_{22}c_1)\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^2\partial\hat{y}} + (\hat{a}_{12}c_3 + \hat{a}_{22}c_2)\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}^2} + \hat{a}_{22}c_3\frac{\partial^3\hat{\phi}_j}{\partial\hat{y}^3}\right]$$
$$= \left(\frac{1}{2|K|}\right)^3 \left[d_{21}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^3} + d_{22}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^2\partial\hat{y}} + d_{23}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}^2} + d_{24}\frac{\partial^3\hat{\phi}_j}{\partial\hat{y}^3}\right].$$

Therefore,

$$\frac{\partial(\Delta\phi_j)}{\partial n} = \left(\frac{1}{2|K|}\right)^3 \left[\left(d_{11}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^3} + d_{12}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^2\partial\hat{y}} + d_{13}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}^2} + d_{14}\frac{\partial^3\hat{\phi}_j}{\partial\hat{y}^3}\right)n_x \right.$$
$$\left. + \left(d_{21}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^3} + d_{22}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}^2\partial\hat{y}} + d_{23}\frac{\partial^3\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}^2} + d_{24}\frac{\partial^3\hat{\phi}_j}{\partial\hat{y}^3}\right)n_y\right]$$

2.8 Numerical Quadrature

2.8.1 Gaussian Quadrature

Numerical quadrature rules utilized in this research were implemented based on standard Gaussian Quadrature available from any standard numerical analysis textbook. Basically, for 1D Gaussian quadrature a set of ordered pairs of abscissas and weights (x_q, w_q) are maintained in the code to obtain various degrees of accuracy based on the polynomial degree $r-1$. This is what is used in computing the edge based line integrals. For 2D, a set of ordered triples (x_q, y_q, w_q) are maintained for various degrees $r-1$ in a similar manner. To compute an approximation to the integral, one has to simply sum the weights multiplied by the integrand evaluated at the abscissas.

2.8.2 Templates

Templates are the term we use for calculations which can be done at the beginning of the run and which do not change during the run. All basis functions are evaluated at all quadrature points on the reference triangle $\hat{K}$ and edges of the reference triangle which reduces both 2-D and 1-D approximation of integrals to simple sums.

In addition, penalty terms involving $\langle\phi_j, \phi_i\rangle_e$ for both within element and between elements along an edge e are evaluated at the beginning and stored in *penalty* and *penalty mix* templates for both `Ell2` and `Ell4`.

2.9 Linear Solvers

The use of the SIPG formulation produces a large sparse, symmetric positive definite linear system which is required to be solved at each adaptive iteration. Coupled with the fact that a natural hierarchy exists in the geometric mesh, there exists a selection of linear solvers that can be used to approximate the solution. Due to the sparse nature of the stiffness matrix, all of these methods are iterative in nature.

While there exists much literature regarding these basic algorithms, there are some excellent treatises which were used quite often throughout this research. For an overall treatment of many

Table 2.9: Linear Solver Support Routines

Routine	Purpose	Algorithm
`mas_ddot`	Dot product: $\alpha = x^{\mathsf{T}} y$	6
`mas_daxpy`	Daxpy: $z \leftarrow x + \alpha y$	7
`mas_dcopy`	Vector copy: $y \leftarrow x$	8
`mas_Dmltav`	Matrix-vector multiplication: $y \leftarrow Ax$	9
`mas_Resid`	Residual calculation: $r \leftarrow b - Ax$	10

Algorithm 6 `mas_ddot`: Dot Product

Require: Composite Mesh Level $\mathcal{L}$, Composite Level Mesh $\mathcal{T}_{\mathcal{L}}$
Require: Vectors x, y
  $\alpha = 0$
  for all $\ell \in [0, \mathcal{L}]$ **do**
    for all $K \in T_{\ell}^{Lf}$ **do**
      $\alpha = \alpha + x_K^{\mathsf{T}} y_K$
    end for
  end for
  for all $K \in T_{\mathcal{L}}^{NLf}$ **do**
    $\alpha = \alpha + x_K^{\mathsf{T}} y_K$
  end for

iterative methods, see Golub and van Loan (1996), Saad (2003), and Greenbaum (1997). For classical iterative methods such as Gauss-Seidel and Jacobi, see Young (1971) and Varga (1962). For more on Multigrid methods and algorithms, see Hackbusch (1985) and Bramble (1993).

2.9.1 Notation and Support Algorithms

In its most basic form, one desires to solve the matrix equation $Ax = b$. We will commonly utilize the residual vector $r = b - Ax$ in our computations. Let $\mathcal{T} := \mathcal{T}_L$ be the current mesh with hierarchy level L. Regarding multi-level methods, we will consider the set of composite level based meshes $\{\mathcal{T}_\ell\}_{\ell=0}^{L}$ as defined in Equation 2.1. There may be on occasion the need for auxiliary or temporary vectors, these will be indicated by p, q and others as required. To indicate a particular iterate of the iterative solver algorithm, we will use a superscript in parentheses, i.e., $x^{(i)}$ would be the x vector at the i^{th} solver iteration.

Integral to any matrix iterative algorithm are basic computational routine which are utilized repeatedly. Because of the sparse nature of the systems to be solved and the specialized data structures that we have implemented, we have developed a number of routines which in essence does for our data objects what BLAS does for dense scaler, vector and matrix operations. These routines are listed in Table 2.9, and the algorithms are listed in Algorithms 6–10.

Note that `mas_Dmltav` and `mas_Resid` both involve multiplication by the global stiffness matrix A. However, this matrix consists of diagonal matrix blocks A_K associated with triangles and off-diagonal matrix blocks A_e associated with interior edges, i.e., the global stiffness matrix is never fully assembled, it is put together "on the fly" and only the *action* on a vector is computed.

In fact, there are two possible instances of A_K which must be maintained for any triangle K which exists on the LEAF of $\mathcal{T}$. Consider a triangle $K \in \mathcal{T}$, LEVEL$(K) = L-1$, and there exists a hanging node on an edge $e \in \partial K$. One can view this situation by considering Figure 2.2(c) with $L = 2$,

Algorithm 7 `mas_daxpy`: Daxpy

Require: Composite Mesh Level $\mathcal{L}$, Composite Level Mesh $\mathcal{T}_{\mathcal{L}}$
Require: Scaler α, Vectors x, y, z
  for all $\ell \in [0, \mathcal{L}]$ **do**
    for all $K \in T_{\ell}^{Lf}$ **do**
      $z_K = \alpha x_K + y_K$
    end for
  end for
  for all $K \in T_{\mathcal{L}}^{NLf}$ **do**
    $z_K = \alpha x_K + y_K$
  end for

Algorithm 8 `mas_dcopy`: Dcopy

Require: Composite Mesh Level $\mathcal{L}$, Composite Level Mesh $\mathcal{T}_{\mathcal{L}}$
Require: Vectors x,y
  for all $\ell \in [0, \mathcal{L}]$ **do**
    for all $K \in T_{\ell}^{Lf}$ **do**
      $y_K = x_K$
    end for
  end for
  for all $K \in T_{\mathcal{L}}^{NLf}$ **do**
    $y_K = x_K$
  end for

Algorithm 9 `mas_Dmltav`: Matrix-Vector Multiplication

Require: Composite Mesh Level $\mathcal{L}$, Composite Level Mesh $\mathcal{T}_{\mathcal{L}}$
Require: Vectors x,y, matrix A
  for all $\ell \in [0, \mathcal{L})$ **do**
    for all $K \in T_{\ell}^{Lf}$ **do**
      $y_K = A_{K,nbrstate} x_K$
    end for
  end for
  for all $K \in T_{\mathcal{L}}$ **do**
    $y_K = A_{K,0} x_K$
  end for
  for all $\ell \in [0, \mathcal{L})$ **do**
    for all $e \in E_{\ell}^{I,Lf}$ **do**
      $y_{K^-} = y_{K^-} + A_e x_{K^+}$; $y_{K^+} = y_{K^+} + A_e^{\mathsf{T}} x_{K^-}$
    end for
  end for
  for all $e \in E_{\mathcal{L}}^{I}$ **do**
    $y_{K^-} = y_{K^-} + A_e x_{K^+}$; $y_{K^+} = y_{K^+} + A_e^{\mathsf{T}} x_{K^-}$
  end for

Algorithm 10 `mas_Resid`: Residual

Require: Composite Mesh Level $\mathcal{L}$, Composite Level Mesh $\mathcal{T}_{\mathcal{L}}$
Require: Vectors x,r, b, matrix A
  for all $\ell \in [0, \mathcal{L})$ **do**
    for all $K \in T_\ell^{Lf}$ **do**
      $r_K = b_K - A_{K,nbrstate} x_K$
    end for
  end for
  for all $K \in T_{\mathcal{L}}$ **do**
    $r_K = b_K - A_{K,0} x_K$
  end for
  for all $\ell \in [0, \mathcal{L})$ **do**
    for all $e \in E_\ell^{I,Lf}$ **do**
      $r_{K^-} = r_{K^-} - A_e x_{K^+}$; $r_{K^+} = r_{K^+} - A_e^{\mathsf{T}} x_{K^-}$
    end for
  end for
  for all $e \in E_{\mathcal{L}}^I$ **do**
    $r_{K^-} = r_{K^-} - A_e x_{K^+}$; $r_{K^+} = r_{K^+} - A_e^{\mathsf{T}} x_{K^-}$
  end for

$K = K(0,1)$. Then matrix-vector multiplication on composite mesh $\mathcal{T}_2$ (and on $\mathcal{T}_1$) will involve a diagonal block matrix A_K which has interaction with two neighboring triangles $K(1,1), K(1,2)$, while matrix multiplication on composite mesh $\mathcal{T}_0$ will involve a different diagonal block matrix which interacts only with $K(0,0)$ (Figure 2.2(a)). To indicate which instance of A_K is used in matrix-vector multiplication, we will use $A_{K,0}$ to indicate interaction of K with its neighbors only through its full edges and $A_{K,1}$ to indicate that K interacts with its neighbors where there exists at least one edge with a hanging node. The instance or state of K is indicated with the `nbrstate` flag in the `TRIANGLE_t` data object (see Figure 2.8), and the appropriate instance of the diagonal block can be represented as $A_{K,nbrstate}$. Finally, only one instance of the off-diagonal block matrix associated with edges $e \in \mathcal{E}^I$ needs to be maintained since each edge uniquely separates two triangles, K^+ and K^-.

2.9.2 Conjugate Gradient

Conjugate Gradient (CG) is the workhorse for numerical analysts iteratively solving a symmetric positive definite linear system of equations. The algorithm used in our implementation is listed in Algorithm 11. Through a sequence of *solver iterations*, the algorithm is guaranteed to converge in N iterations if one were working in exact arithmetic. In practice the algorithm achieves an error level close to machine precision.

Note that when CG is starts, the previous solution u obtained from the previous adaptive iteration is embedded into the current finite dimensional space. This utilizes routine `Embed_Prev_Soln` which relies on the embedding operators utilized in Multigrid to embed the solution on the parent triangle K into its four children in the case that K was just refined. In the case where K has not been refined in the previous adaptive iteration, the embedding operator is simply the identity. Were coarsening also being employed, the projection operator from Multigrid would also be utilized. The previous solution is maintained in the vector s.

Algorithm 11 Conjugate Gradient (CG)

Require: Mesh $\mathcal{T}_L$, solver tolerance ϵ_s, Vectors s, u, b, r, p, q, matrix A, total dof N

```
if L > 0 then
  u ← Embed_Prev_Soln
else
  u ← 0
end if
r ← b − Au
for (sIter = 1;sIter ≤ itmax;sIter++) do
  ρ1 = r^T r; err_loc = (ρ1/N)^{1/2}
  if err_loc < ε_s then
    break
  end if
  if sIter = 1 then
    p ← r
  else
    β = ρ1/ρ2; p ← βp + r
  end if
  q ← Ap; d = p^T q
  if d ≠ 0. then
    α = ρ1/d
  end if
  u ← αp + u; r ← r − αq; ρ2 = ρ1
end for
```

In most of the runs presented in this dissertation, the solver tolerance ϵ_s was set to either 1.0e-12 or 1.0e-13. In addition, a maximum number of solver iterations $itmax$ primarily as a debugging aide. Once this tolerance has been reached, CG is finished and the solution is returned in the vector u.

2.9.3 Multigrid

Multigrid is a powerful yet complex algorithm which provides for solutions of linear equations to be obtained in a time proportional to the number N of unknowns in the linear system. The basic idea is to combine a recursive series of coarse grid corrections to the error problem $Ae = r$ which effectively reduces high frequency components of the error coupled with efficient smoothing of the low frequency components of the error to obtain a better approximation to the solution. For any Multigrid algorithm to work, there are some necessary algorithms which must be incorporated including projection and embedding operators and some sort of smoothing algorithm. These are described below. At the end of this section, the Multigrid algorithm utilized here is explained in more detail.

Projection and Embedding Operators

To calculate the projection and embedding operators[11] some discussion is required. It is important to note that there are two different types of operators which transfer data between finite dimensional subspaces. Embedding is defined as moving from a lower dimensional subspace into a higher dimensional subspace. Projection is defined as moving from a higher dimensional subspace into a lower dimensional subspace. We will treat each of these separately.

Let I_H^h denote the embedding operator from the coarse (lower dimension) subspace to the fine (higher dimension) subspace. Similarly, let I_h^H denote the projection operator from the fine subspace to the coarse subspace. We will choose $I_h^H = \left(I_H^h\right)^{\mathsf{T}}$ as the relation between the embedding and projection operators.[12] Finally, we will deal with two different sets of operators. The first set has to do with h, i.e., spatial embedding and projection with the same number of degrees of freedom. The second set has to do with p embedding and projection, where the spatial characteristics are the same but the number of degrees of freedom changes.

h Embedding and Projection. Let $\{\mathbf{x}_j\},\ j = 0, \ldots, n-1$ denote the (x, y) coordinates of the n degrees of freedom on the reference element $\hat{K}$ and $\{\phi_j\},\ j = 0, \ldots, n-1$ denote the corresponding basis functions on the reference element $\hat{K}$. Let

$$\{\mathbf{x}_{k,i}\},\ k = 0, \ldots 3,\ i = 0, \ldots, n-1$$

denote the (x, y) coordinates of the n degrees of freedom on the four children of $\hat{K}$, $\{\hat{K}_0, \hat{K}_1, \hat{K}_2, \hat{K}_3\}$ and $\{\phi_{k,i}\}$ $k = 0, \ldots, 3,\ i = 0, \ldots, n-1$ denote the corresponding basis functions on the four children of $\hat{K}$.

[11] Embedding operators are also called *interpolation* or *prolongation* operators, Projection operators are also called *restriction* operators.

[12] Often one will have $I_h^H = c\left(I_H^h\right)^{\mathsf{T}}$ to indicate weighting of the projection into the coarse subspace. We will work here with $c = 1$.

We define $M_j : \mathbb{R}^2 \to \mathbb{R}^2, \quad j = 0, \ldots, 3$ as

$$M_k \mathbf{x}_i = B_k \mathbf{x}_i + \mathbf{b}_k = \mathbf{x}_{k,i} \quad k = 0, \ldots, 3, \ i = 0, \ldots, n-1 \tag{2.7}$$

where scaling matrix B and translation vector $\mathbf{b}$ are

$$B = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \\ 0.5 & 0 \\ 0 & 0.5 \\ 0.5 & 0 \\ 0 & 0.5 \\ -0.5 & 0 \\ 0 & -0.5 \end{pmatrix} = \begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0.5 \\ 0 \\ 0 \\ 0.5 \\ 0 \\ 0 \\ 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} \tag{2.8}$$

This defines a set of affine transformations of the degrees of freedom on the reference element $\hat{K}$ to the degrees of freedom on the children reference elements $\hat{K}_k \quad k = 0, \ldots, 3$. Therefore one can calculate the coordinates for each degree of freedom for each child of $\hat{K}$ and store them in the matrix C as follows:

$$C = \begin{pmatrix} M_0\mathbf{x}_0 & M_0\mathbf{x}_1 & \cdots & M_0\mathbf{x}_{n-1} \\ M_1\mathbf{x}_0 & M_1\mathbf{x}_1 & \cdots & M_1\mathbf{x}_{n-1} \\ M_2\mathbf{x}_0 & M_2\mathbf{x}_1 & \cdots & M_2\mathbf{x}_{n-1} \\ M_3\mathbf{x}_0 & M_3\mathbf{x}_1 & \cdots & M_3\mathbf{x}_{n-1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{0,0} & \mathbf{x}_{0,1} & \cdots & \mathbf{x}_{0,n-1} \\ \mathbf{x}_{1,0} & \mathbf{x}_{1,1} & \cdots & \mathbf{x}_{1,n-1} \\ \mathbf{x}_{2,0} & \mathbf{x}_{2,1} & \cdots & \mathbf{x}_{2,n-1} \\ \mathbf{x}_{3,0} & \mathbf{x}_{3,1} & \cdots & \mathbf{x}_{3,n-1} \end{pmatrix}$$

$$= \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,n-1} \\ y_{0,0} & y_{0,1} & \cdots & y_{0,n-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,n-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,n-1} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,n-1} \\ y_{2,0} & y_{2,1} & \cdots & y_{2,n-1} \\ x_{3,0} & x_{3,1} & \cdots & x_{3,n-1} \\ y_{3,0} & y_{3,1} & \cdots & y_{3,n-1} \end{pmatrix}$$

We will require that the embedding operator I_H^h simply be the identity for any $u \in V_H$, i.e., $I_H^h u = v$. Thus, we must have

$$u(\mathbf{x}) = \sum_{j=0}^{n-1} \alpha_j \phi_j(\mathbf{x}) = \sum_{k=0}^{3} \sum_{i=0}^{n-1} \alpha_{k,i} \phi_{k,i}(\mathbf{x}).$$

Evaluating $u(\mathbf{x})$ at each coordinate $\{\mathbf{x}_{k,i}\},\ k = 0, \ldots, 3,\ i = 0, \ldots, n-1$ gives us

$$\begin{pmatrix} u(\mathbf{x}_{0,0}) \\ u(\mathbf{x}_{0,1}) \\ \vdots \\ u(\mathbf{x}_{0,n-1}) \\ u(\mathbf{x}_{1,0}) \\ u(\mathbf{x}_{1,1}) \\ \vdots \\ u(\mathbf{x}_{1,n-1}) \\ u(\mathbf{x}_{2,0}) \\ u(\mathbf{x}_{2,1}) \\ \vdots \\ u(\mathbf{x}_{2,n-1}) \\ u(\mathbf{x}_{3,0}) \\ u(\mathbf{x}_{3,1}) \\ \vdots \\ u(\mathbf{x}_{3,n-1}) \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} \\ \alpha_{0,1} \\ \vdots \\ \alpha_{0,n-1} \\ \alpha_{1,0} \\ \alpha_{1,1} \\ \vdots \\ \alpha_{1,n-1} \\ \alpha_{2,0} \\ \alpha_{2,1} \\ \vdots \\ \alpha_{2,n-1} \\ \alpha_{3,0} \\ \alpha_{3,1} \\ \vdots \\ \alpha_{3,n-1}. \end{pmatrix} \tag{2.9}$$

Thus we can define the embedding operator I_H^h as a $4n \times n$ coefficient matrix:

$$I_H^h = \begin{pmatrix} \phi_0(\mathbf{x}_{0,0}) & \phi_1(\mathbf{x}_{0,0}) & \cdots & \phi_{n-1}(\mathbf{x}_{0,0}) \\ \phi_0(\mathbf{x}_{0,1}) & \phi_1(\mathbf{x}_{0,1}) & \cdots & \phi_{n-1}(\mathbf{x}_{0,1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_{0,n-1}) & \phi_1(\mathbf{x}_{0,n-1}) & \cdots & \phi_{n-1}(\mathbf{x}_{0,n-1}) \\ \phi_0(\mathbf{x}_{1,0}) & \phi_1(\mathbf{x}_{1,0}) & \cdots & \phi_{n-1}(\mathbf{x}_{1,0}) \\ \phi_0(\mathbf{x}_{1,1}) & \phi_1(\mathbf{x}_{1,1}) & \cdots & \phi_{n-1}(\mathbf{x}_{1,1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_{1,n-1}) & \phi_1(\mathbf{x}_{1,n-1}) & \cdots & \phi_{n-1}(\mathbf{x}_{1,n-1}) \\ \phi_0(\mathbf{x}_{2,0}) & \phi_1(\mathbf{x}_{2,0}) & \cdots & \phi_{n-1}(\mathbf{x}_{2,0}) \\ \phi_0(\mathbf{x}_{2,1}) & \phi_1(\mathbf{x}_{2,1}) & \cdots & \phi_{n-1}(\mathbf{x}_{2,1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_{2,n-1}) & \phi_1(\mathbf{x}_{2,n-1}) & \cdots & \phi_{n-1}(\mathbf{x}_{2,n-1}) \\ \phi_0(\mathbf{x}_{3,0}) & \phi_1(\mathbf{x}_{3,0}) & \cdots & \phi_{n-1}(\mathbf{x}_{3,0}) \\ \phi_0(\mathbf{x}_{3,1}) & \phi_1(\mathbf{x}_{3,1}) & \cdots & \phi_{n-1}(\mathbf{x}_{3,1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_{3,n-1}) & \phi_1(\mathbf{x}_{3,n-1}) & \cdots & \phi_{n-1}(\mathbf{x}_{3,n-1}) \end{pmatrix}$$

or

$$I_H^h = \begin{pmatrix} \phi_0(\hat{C}) & \phi_1(\hat{C}) & \cdots & \phi_{n-1}(\hat{C}) \end{pmatrix}$$

where $\hat{C} \in \mathbb{R}^{4n}$ represents C transformed to a column vector and which maps a vector $u \in \mathbb{R}^n$ into a vector $v \in \mathbb{R}^{4n}$. The projection operator is $I_h^H = (I_H^h)^\mathsf{T}$.

p **Embedding and Projection.** We can proceed along a similar path for determining the embedding from V_h^r to V_h^s, $s = r+1, \ldots, 5$, $r = 2, 3, 4$. Note that as before we will work on the reference element $\hat{K}$. Let the higher dimensional subspace V_h^s contain m degrees of freedom, $m > n$.

$$\{\mathbf{x}_{s,i}\}, i = 0, \ldots, m_1 - 1$$

denote the (x, y) coordinates of the m degrees of freedom of the higher dimensional subspace on $\hat{K}$ and $\{\phi_{s,i}\}$ $i = 0, \ldots, m-1$ denote the corresponding basis functions on the same subspace. We will require that the embedding operator I_r^s simply be the identity for any $u \in V_h^r$, i.e., $I_r^s u = v$. Thus, we must have

$$u(\mathbf{x}) = \sum_{j=0}^{n-1} \alpha_j \phi_j(\mathbf{x}) = \sum_{i=0}^{m-1} \alpha_{s,i} \phi_{s,i}(\mathbf{x}).$$

Evaluating $u(\mathbf{x})$ at each coordinate $\{\mathbf{x}_{s,i}\}$, $i = 0, \ldots, m-1$ gives us

$$\begin{pmatrix} u(\mathbf{x}_{s,0}) \\ u(\mathbf{x}_{s,1}) \\ \vdots \\ u(\mathbf{x}_{s,m-1}) \end{pmatrix} = \begin{pmatrix} \alpha_{s,0} \\ \alpha_{s,1} \\ \vdots \\ \alpha_{s,m-1}. \end{pmatrix} \tag{2.10}$$

Thus we can define the embedding operator I_r^s as a $m \times n$ coefficient matrix:

$$I_r^s = \begin{pmatrix} \phi_0(\mathbf{x}_{s,0}) & \phi_1(\mathbf{x}_{s,0}) & \cdots & \phi_{n-1}(\mathbf{x}_{s,0}) \\ \phi_0(\mathbf{x}_{s,1}) & \phi_1(\mathbf{x}_{s,1}) & \cdots & \phi_{n-1}(\mathbf{x}_{s,1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}_{s,m-1}) & \phi_1(\mathbf{x}_{s,m-1}) & \cdots & \phi_{n-1}(\mathbf{x}_{s,m-1}). \end{pmatrix}$$

We can then define the projection operator $I_s^r = (I_r^s)^\mathsf{T}$.

Smoothing

We utilize a *block* version of Gauss-Seidel iteration for our smoother. Note that by *block* we simply mean that we are updating the x vector on a triangle by triangle basis. On each particular triangle K, we update x_K as one would normally do with the classical or point Gauss-Seidel, i.e.,

$$x_{K,i}^k = \frac{1}{a_{K,ii}} \left(b_{K,i} - \sum_{j<i} a_{ij} x_j^k - \sum_{j>i} a_{ij} x_j^{k+1} \right)$$

where the index j ranges over all triangles interacting with triangle K through K's edges. There will be additional variants of this version of the smoother introduced in § 2.10.2. Our Gauss-Seidel smoother is shown in Algorithm 12.

The Multigrid Algorithm

We are now ready to discuss in more detailed our implementation of Multigrid. Algorithm 13 illustrates the main Multigrid solver routine and Algorithm 14 illustrates the recursive Multigrid cycle algorithm. If one chooses $\mu = 1$ one obtains a V-cycle Multigrid scheme, if $\mu = 2$ one obtains

Algorithm 12 Smoother 0: `GS_Block`

Require: Composite Mesh Level: $\mathcal{L}$, Mesh Hierarchy: $\mathcal{T}_{\mathcal{L}}$, Number of smoothings: ν
Require: Vectors x, y, Matrix A
  Calculate ℓ_0 for global/local smoothing
  for $(i = 0; i < \nu; i++)$ **do**
    for $(\ell = \ell_0; \ell < \mathcal{L}; \ell++)$ **do**
      for all $K \in T_\ell^{Lf}$ **do**
        $x_K \leftarrow$ Updated over all DOF using point GS
      end for
    end for
    for all $K \in T_{\mathcal{L}}$ **do**
      $x_K \leftarrow$ Updated over all DOF using point GS
    end for
  end for

Algorithm 13 Multigrid (MG)

Require: Mesh Hierarchy $\mathbb{T}$, solver tolerance ϵ_s, ν_1, ν_2, μ
Require: Vectors s, u, b, r, p, q, matrix A, total dof N
  if $L > 0$ **then**
    $u \leftarrow$ `Embed_Prev_Soln`
  else
    $u \leftarrow 0$
  end if
  for $(sIter = 1; sIter \leq itmax; sIter++)$ **do**
    $u \leftarrow$`MGCYC`$(L, \nu_1, \nu_2, \mu, u_L, b_L, r_L)$
    $r \leftarrow b - Au$
    $\rho = r^{\mathsf{T}} r$
    $err_{loc} = \sqrt{\rho/N}$
    if $err_{loc} < \epsilon_s$ **then**
      `break`
    end if
  end for

Algorithm 14 Multigrid Cycle (MGCYC)

Require: Mesh level $\mathcal{L}$, Composite Level Mesh $\mathcal{T}_{\mathcal{L}}$, ν_1, ν_2, μ
Require: Vectors x, y, z, matrix A
  if $\mathcal{L} < L$ **then**
    for all $\ell \in [0, \mathcal{L}]$ **do**
      for all $K \in T_\ell^{Lf}$ **do**
        $x_{K,\mathcal{L}+1} \leftarrow x_{K,\mathcal{L}}$; $x_{K,\mathcal{L}} \leftarrow 0$; $y_{K,\mathcal{L}+1} \leftarrow y_{K,\mathcal{L}}$; $y_{K,\mathcal{L}} \leftarrow z_{K,\mathcal{L}}$
      end for
    end for
    for all $K \in T_{\mathcal{L}}^{NLf}$ **do**
      $x_{K,\mathcal{L}} \leftarrow 0$; $y_{K,\mathcal{L}} \leftarrow z_{K,\mathcal{L}}$
    end for
  end if
  if $\mathcal{L} = 0$ **then**
    $x_0 \leftarrow$ `Coarse_Solve`(x_0, y_0, A_0)
  else
    $x_{\mathcal{L}} \leftarrow S^{\nu_1}(x_{\mathcal{L}}, y_{\mathcal{L}}, A_{\mathcal{L}})$
    $z_{\mathcal{L}} \leftarrow y_{\mathcal{L}} - A_{\mathcal{L}} x_{\mathcal{L}}$
    $z_{\mathcal{L}-1} \leftarrow I_{\mathcal{L}}^{\mathcal{L}-1} z_{\mathcal{L}}$
    for $(i = 1, \mu)$ **do**
      $x_{\mathcal{L}-1} \leftarrow$`MGCYC`$(\mathcal{L}-1, \nu_1, \nu_2, \mu, x_{\mathcal{L}-1}, y_{\mathcal{L}-1}, z_{\mathcal{L}-1})$
    end for
    $x_{\mathcal{L}} \leftarrow I_{\mathcal{L}-1}^{\mathcal{L}} x_{\mathcal{L}-1}$
    $x_{\mathcal{L}} \leftarrow S^{\nu_2}(x_{\mathcal{L}}, y_{\mathcal{L}}, A_{\mathcal{L}})$
  end if
  if $\mathcal{L} < L$ **then**
    for all $\ell \in [0, \mathcal{L}]$ **do**
      for all $K \in T_\ell^{Lf}$ **do**
        $x_{K,\mathcal{L}+1} \leftarrow x_{K,\mathcal{L}} + x_{K,\mathcal{L}+1}$; $y_{K,\mathcal{L}} \leftarrow y_{K,\mathcal{L}+1}$
      end for
    end for
  end if

a W-cycle Multigrid scheme. The number of pre-smoothings performed is ν_1 and the number of post-smoothings performed is ν_2.

The exact solve performed on the coarsest hierarchy level is performed using the `clapack_dposv` and `clapack_dpotrs` routines from `ATLAS` package (see 2.11.3). Note the the cholesky factorization is performed once on the full level 0 stiffness matrix (including the presence of 0.0's), and that future use of the exact solve is simply a backward-forward substitution process.

A couple of important notes are in order here. First, due to the manner in which we are storing our vectors, we maintain a separate (u_K, b_K) pair for each possible level in the hierarchy. In the MGCYC routine we save the previous values upon entrance and restore the values upon exit of the routine. This allows one to work with "working" vectors which simplifies the algorithms used throughout Multigrid and one does not have to worry about overwriting values which will be required when one returns from the recursive calls. Second, we embed the previous solution obtained into the current mesh so as to obtain a better starting approximation. Third, for implementation of the Multigrid algorithms for the biharmonic test problems, we utilize a variable number of smoothings, increasing the number of smoothings on each successive coarser level. This appeared to help with the number of solver iterations required to reach the solver tolerance. Finally, for all of the runs referenced in this research, we utilize $\mu = 1$, $\nu_1 = \nu_2 = 4$ when using Multigrid as a solver.

2.9.4 Preconditioned Conjugate Gradient

Our implementation of Preconditioned Conjugate Gradient (PCG) utilizes Multigrid as a preconditioner. Our implementation of PCG is shown in Algorithm 15. When we are using MG as a preconditioner, we utilize $\mu = 1$, $\nu_1 = \nu_2 = 2$. In addition, we ensure that MG is a symmetric preconditioner in that during the pre and post smoothing actions, we utilize forward Gauss-Seidel as the pre smoother and backward Gauss-Seidel as the post smoother. This is rather easy to implement as one just visits the elements and unknowns in reverse order.

2.9.5 Solver Comparison

Some comparison charts presenting a comparison of the different solvers for some selected test cases are presented in Figures 2.21–2.28. In this comparison we look at CG, MG, and PCG.

There are some observations which must be made here. For the multilevel algorithms MG and PCG, the number of solver iterations becomes essentially constant. Referring to Figure 2.21(b) one can see that PCG reduces the residual approximately by an average multiple of 0.25 each adaptive iteration, while CG approaches 1.0 asymptotically, i.e., not much average reduction in residual occurs per solver iteration, i.e., it is not very efficient. Looking at Figure 2.22(a) one can see that PCG takes less time to solve for the unknown per dof. Finally, one should note that in Figure 2.22(b) the initial high MFLOP/s rate for MG and PCG is due to performing an exact solve on the very first multilevel iteration using the optimized BLAS from ATLAS linked with lapack factoring and solve routines. Overall it is clear that PCG and MG are very good solvers, tweaking the various parameters might provide even better performance.

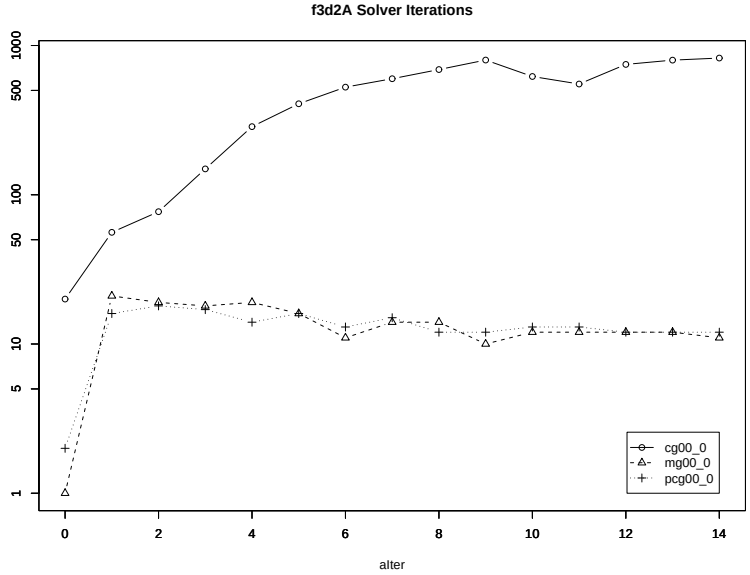
2.10 Performance Optimization and Monitoring

2.10.1 Background

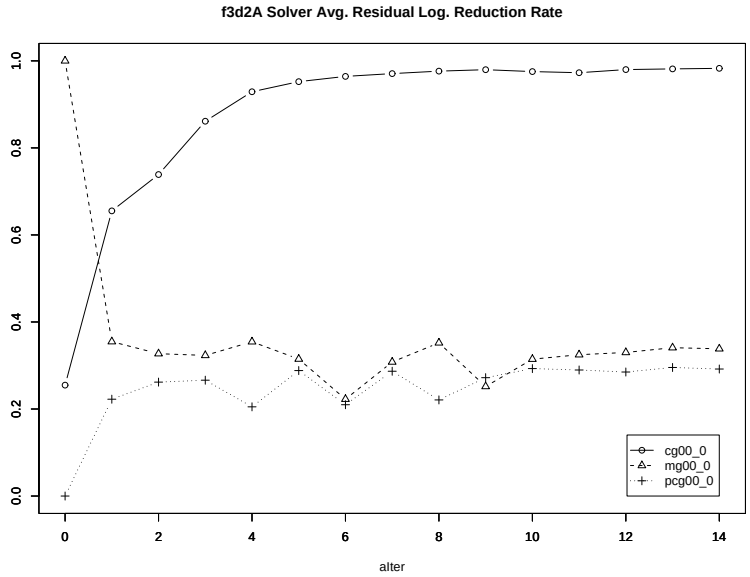
One of the goals of this research was to develop an adaptive DG FEM program which runs quickly. In order to achieve this there are three areas which we focused on. The first area was that of compiler

Algorithm 15 Preconditioned Conjugate Gradient (PCG)

Require: Mesh $\mathcal{T}_L$, solver tolerance ϵ_s, ν_1, ν_2, μ
Require: Vectors s, u, b, r, p, q, v, z, matrix A, total dof N
  if $L > 0$ **then**
    $u \leftarrow$ `Embed_Prev_Soln`
  else
    $u \leftarrow 0$
  end if
  $r \leftarrow b - Au$
  for $(sIter = 1; sIter \leq itmax; sIter++)$ **do**
    $z \leftarrow 0$
    $z \leftarrow$`MGCYC`$(L, \nu_1, \nu_2, \mu, z, r, v)$
    $\rho_1 = r^{\mathsf{T}} r$; $\rho_{z1} = z^{\mathsf{T}} r$; $err_{loc} = (\rho_1/N)^{1/2}$
    if $err_{loc} < \epsilon_s$ **then**
      `break`
    end if
    if $sIter = 1$ **then**
      $p \leftarrow z$
    else
      $\beta = \rho_{z1}/\rho_{z2}$; $p \leftarrow \beta p + z$
    end if
    $q \leftarrow Ap$; $d = p^{\mathsf{T}} q$
    if $d \neq 0.$ **then**
      $\alpha = \rho_{z1}/d$
    end if
    $u \leftarrow \alpha p + u$; $r \leftarrow r - \alpha q$; $\rho_{z2} = \rho_{z1}$
  end for

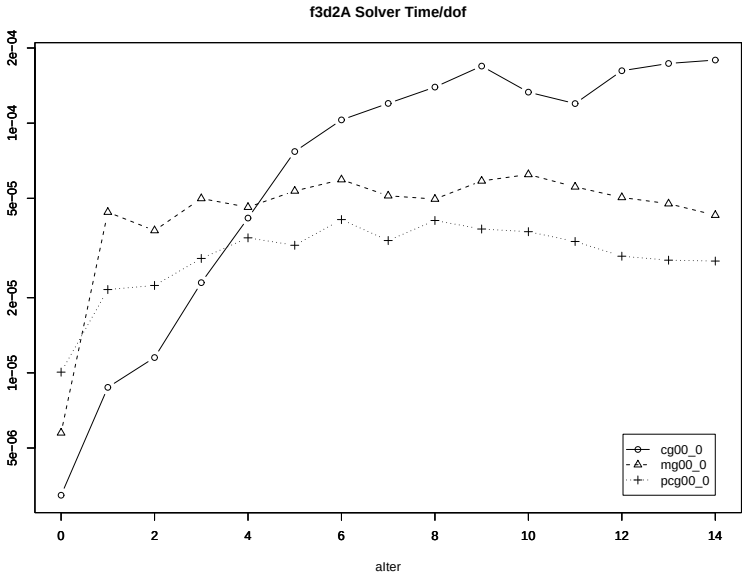


(a) Solver Iterations

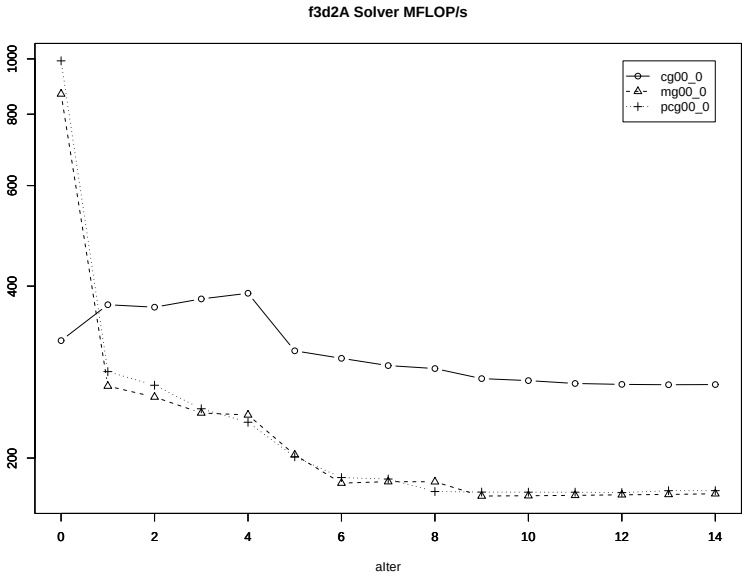


(b) Solver Avg.Log. Residual Reduction Rate

Figure 2.21: Solver Comparison (Efficiency): f3, $r = 3$, Adaptive

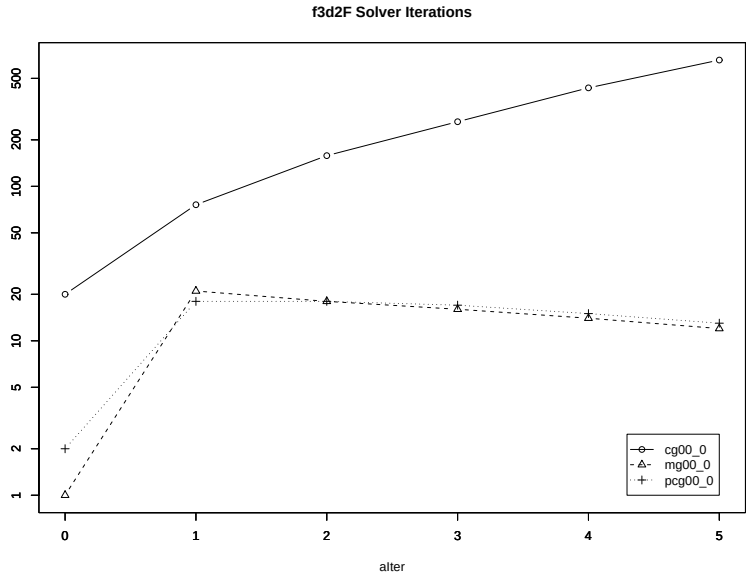


(a) Solver Time/dof

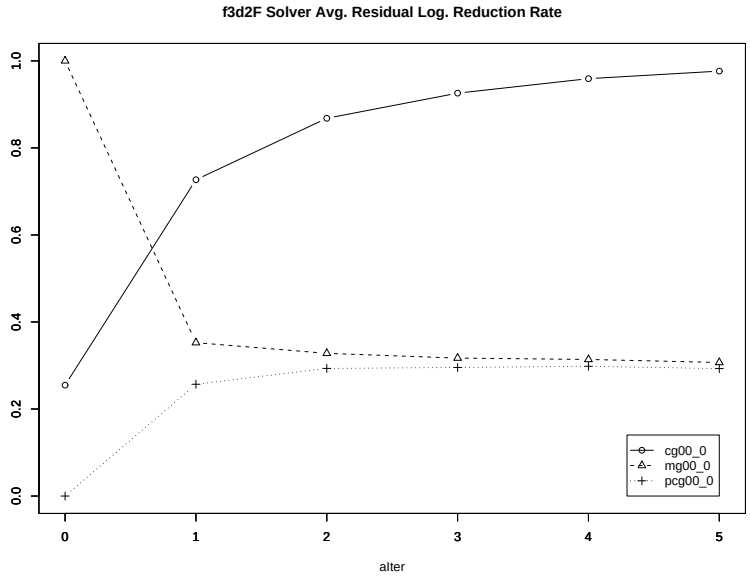


(b) Solver MFLOP/s

Figure 2.22: Solver Comparison (Timing): f3, $r = 3$, Adaptive

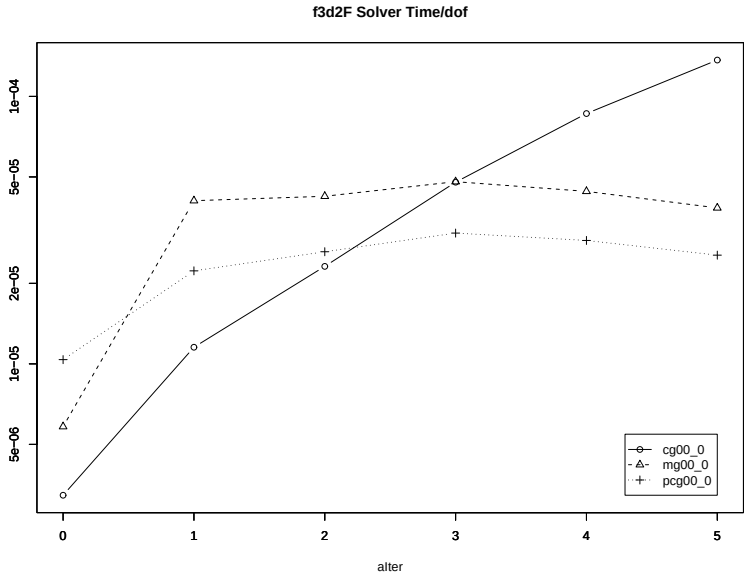


(a) Solver Iterations

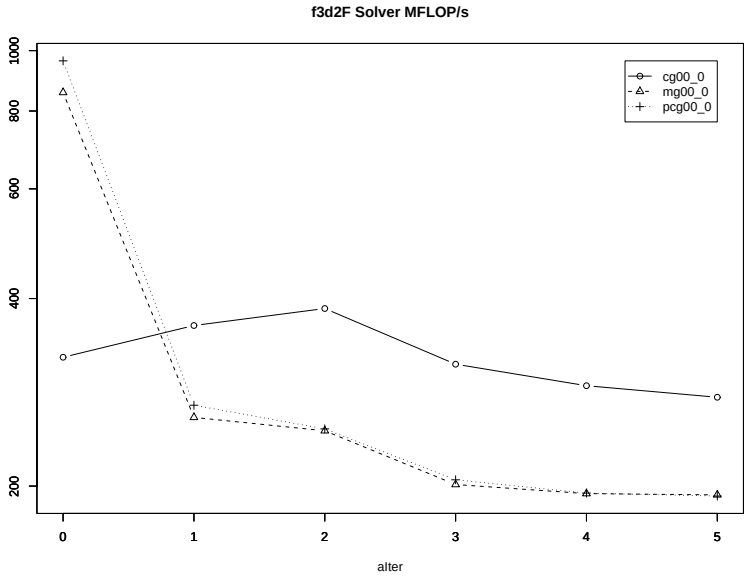


(b) Solver Avg.Log. Residual Reduction Rate

Figure 2.23: Solver Comparison (Efficiency): f3, $r = 3$, Uniform

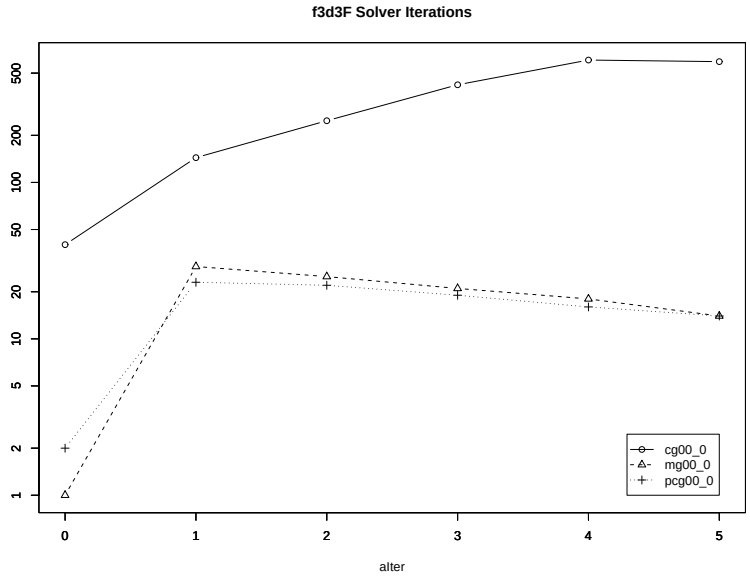


(a) Solver Time/dof

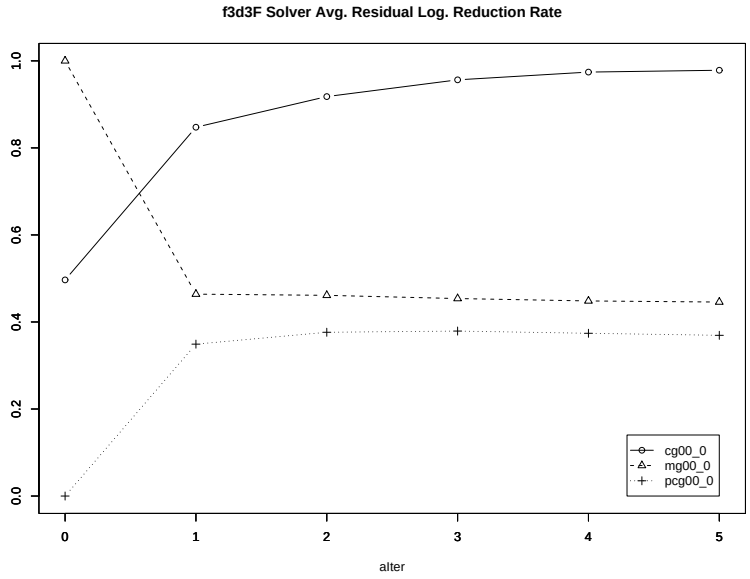


(b) Solver MFLOP/s

Figure 2.24: Solver Comparison (Timing): f3, $r = 3$, Uniform

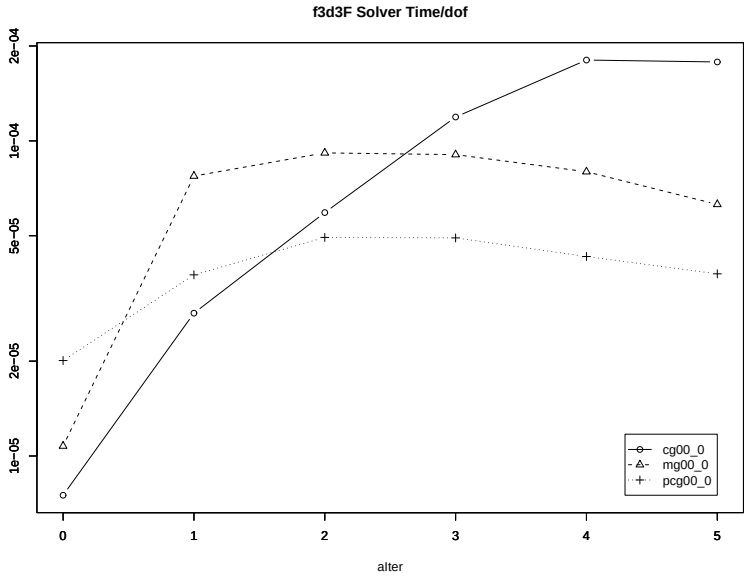


(a) Solver Iterations

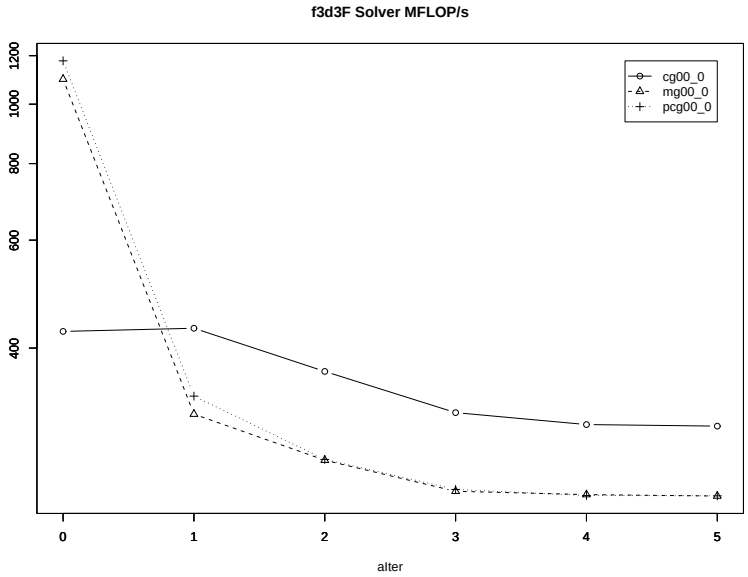


(b) Solver Avg.Log. Residual Reduction Rate

Figure 2.25: Solver Comparison (Efficiency): f3, $r = 4$, Uniform

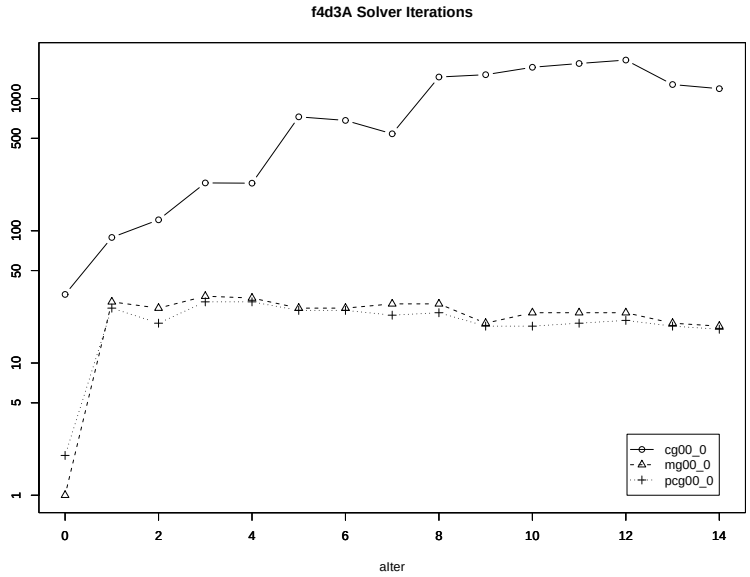


(a) Solver Time/dof

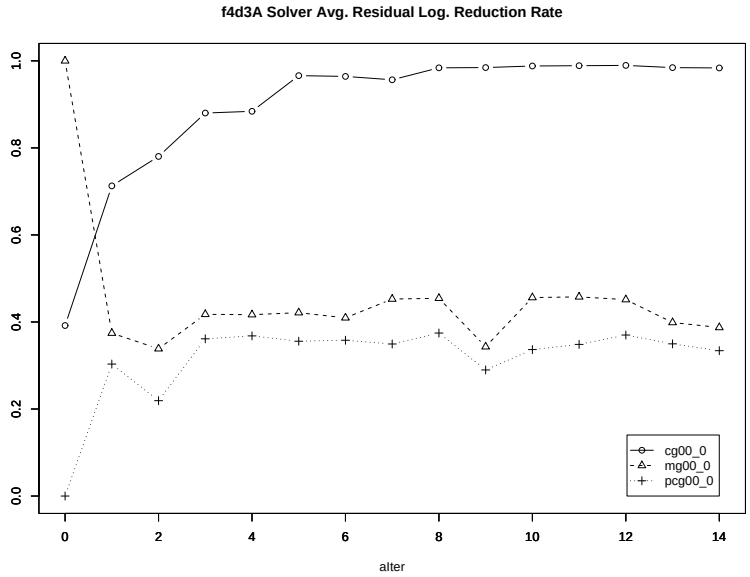


(b) Solver MFLOP/s

Figure 2.26: Solver Comparison (Timing): f3, $r = 4$, Uniform

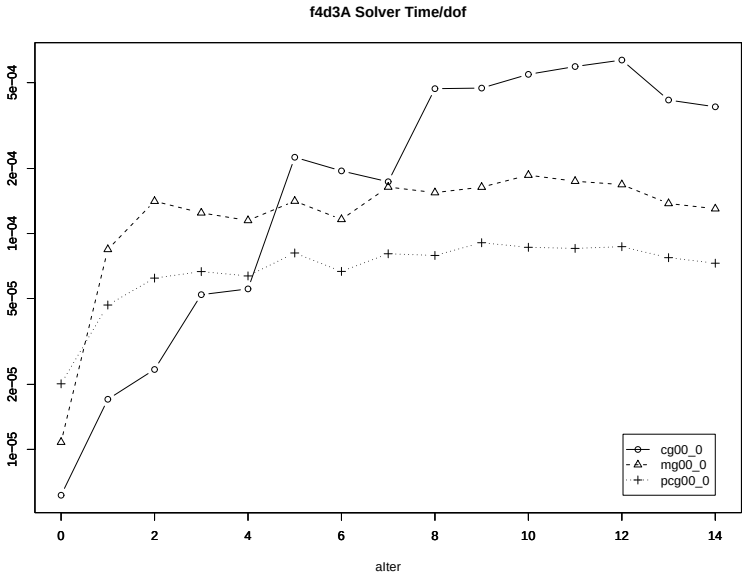


(a) Solver Iterations

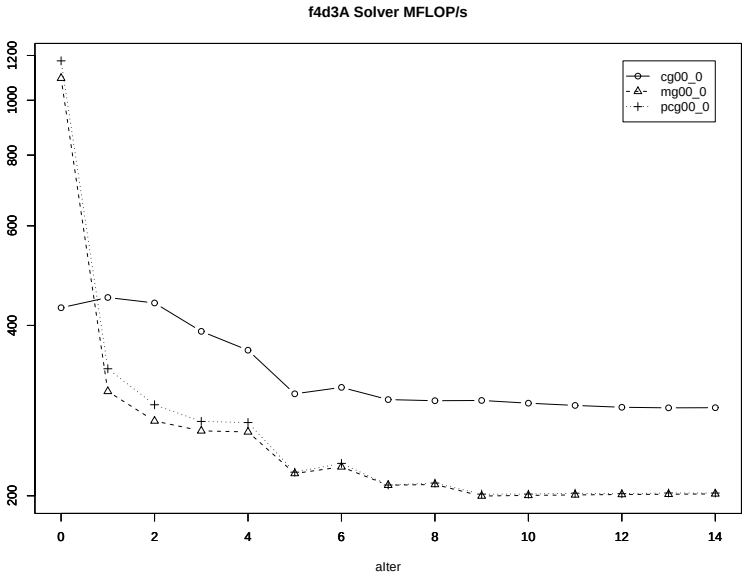


(b) Solver Avg.Log. Residual Reduction Rate

Figure 2.27: Solver Comparison (Efficiency): f4, $r = 4$, Adaptive



(a) Solver Time/dof



(b) Solver MFLOP/s

Figure 2.28: Solver Comparison (Timing): f4, $r = 4$, Adaptive

choice and optimization flags. We utilized primarily the `gnu` compiler collection during the course of this research. For the machines we were running on (`Intel Pentium4`), we found that the following choice of compiler flags performed the basic optimizations one would want from a compiler:

```
-O3 -march=pentium4 -mcpu=pentium4 -msse -msse2 -mfpmath=sse -malign-double
 -funroll-loops -finline-functions.
```

Thus, we did not attempt to do much "manual" performance optimization tuning. For more on those techniques, see Goedecker and Hoisie (2001), Weiss (2001), Beyls (2004), and Kowarschik (2004). Figure 2.29 shows comparison timings for some selected compiler flags and choices. Note that in Figure 2.29 the following notation is used:

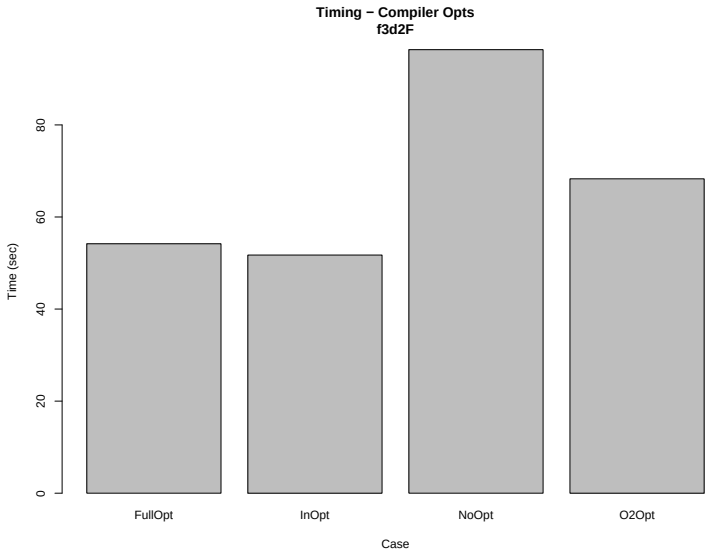
- NoOpt: `gcc -O0` - No Optimization
- O2Opt: `gcc -O2` - Medium Optimization
- FullOpt: `gcc` - Aggressive Optimization
- InOpt: `icc` - Aggressive Optimization (Intel).

The second area is based on the ideas put forth by Douglas et al. (2000) termed *cache blocking*. The basic idea applies to an algorithm such as Gauss-Seidel which is required to be repeated a fixed number of times, i.e., Gauss-Seidel Smoothing inside of the Multigrid method. I elaborate on how these ideas can be applied to our situation in § 2.10.2. I should note that the Douglas cache blocking scheme is only applied to the composite level mesh on the current maximum level of the hierarchy L, and differs from his implementation in that I do not attempt to apply cache blocking when working with composite level meshes on levels $\ell < L$ in a Multigrid context. This seemed like a reasonable approach since the majority of the work is done on the full Leaf. Note that we did not perform all of the optimizations that Douglas identified, therefore we did not include the residual calculation inside of the cache blocked Gauss-Seidel as mentioned in Douglas et al. (2000).

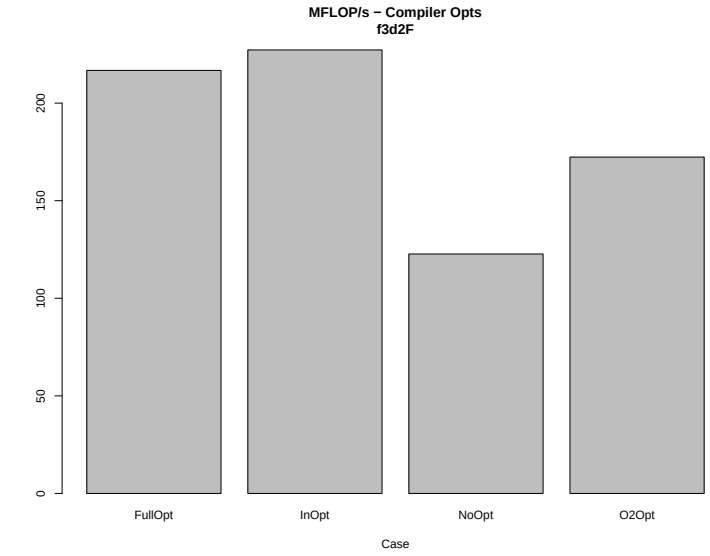
The last area in which performance gains can be realized has to do with the organizational layout of the stiffness matrix in memory. Recall that these block matrices have some data contiguity based on the manner in which the elements are managed. To test further we allocated large chunks of memory to store "working" copies of the complete stiffness matrix in a *Modified Block Sparse Row* format (Saad, 2003) on the Leaf lists and on the Leaf and Non-Leaf lists for each level in the hierarchy. This basically doubles the storage required for a run, however results are promising and are presented below. One are of future research will be to convert to an object management strategy which combines the best of the list based approach with the MBSR storage format for performing the matrix calculations.

2.10.2 Cache Blocking

Suppose that one is given a mesh $\mathcal{T}$ (which implies all of the above partitions exist). Associated with this mesh are $N_\tau := |\mathcal{T}|$ triangles. Since the majority of computations are $\mathcal{T}$ based iteration, it makes sense to try and reuse data in cache as much as possible, since for large meshes $\mathcal{T}$ will not exist completely in cache. This strategy is most applicable to routines for which repetitive calculations can be be broken up into blocks which can be computed in parallel, i.e., little or no intercommunication is required. It is fortunate that Multigrid using Gauss-Seidel as a smoother can be adapted in this manner in a number of ways. So, following primarily the work of Douglas et al. (2000) (see also Strout et al. (2001) for another variant), then one must estimate accurately the size of the data used in the computations and partition $\mathcal{T}$ in an advantageous manner so that cache reutilization (primarily L2) is maximized; this is often called *cache blocking* or *cache tiling*.



(a) Time (sec)



(b) MFLOP/s

Figure 2.29: Optimization Flags/Compiler Comparison : f3, $r = 3$, Uniform

Assume that the cache sizing calculations determine that C triangles and their associated required data will reside completely in a percentage of the total L2 cache[13] Thus, the number of cache blocks for $\mathcal{T}$ would be $N_b := \max\left(\lfloor \frac{N_\tau}{C} \rfloor, 1\right)$. For now, let N_c be the fixed number of sub-blocks or zones [14] which subdivide each cache block. This leads to the following collection of subsets of $\mathcal{T}$:

$$
\begin{aligned}
\mathcal{S} &:= \{S_b\}_{b=1}^{N_b} = \{K \in \mathcal{T}\} \\
S_b &:= \bigoplus_{\ell=0}^{L} S_b^\ell = \{K \in \mathcal{T} | \text{ cache block } b\} \\
S_b^\ell &:= \bigoplus_{c=1}^{N_c} S_{bc}^\ell = \{K \in T_\ell^{Lf} | \text{ cache block } b\} \\
S_{bc}^\ell &:= \{K \in T_\ell^{Lf} | \text{ cache block } b, \text{ cache sub-block } c\} \\
S_b^{\ell,N} &:= \bigoplus_{c=1}^{N_c} S_{bc}^{\ell,N} = \{K \in T_\ell^{NLf} | \text{ cache block } b\} \\
S_{bc}^{\ell,N} &:= \{K \in T_\ell^{NLf} | \text{ cache block } b, \text{ cache sub-block } c\}.
\end{aligned}
$$

The basic idea for cache blocking of Gauss-Seidel is to partition the domain into blocks, and partition each block into sub-blocks, the first sub-block which is the block-block boundary of each block. We utilize the `METIS` (see 2.11.7 and Karypis and Kumar (1995)) software to perform this task. We then divide each block into sub-blocks numbering in increasing order the sub-blocks from the block boundary toward the interior. The number of sub-blocks is determined based on the fixed number of repetitions required of the Gauss-Seidel algorithm. The result of this process is illustrated in Figure 2.30 for $\mathcal{T}$ with $N_b = 4, N_c = 4, N_s = 3$.

One area of future work is to utilize more sophisticated domain decomposition methods to obtain the solution of the linear systems. The basic process of decomposing the domain has been done. One area which was explored was utilization of overlapping the blocks along the block boundaries. This method showed some promise and will be explored further at a later date.

Note also the following:

$$
T_b = \bigcup_{c=1}^{N_c} T_{bc}, \quad b = 1, \ldots, N_b \tag{2.11}
$$

$$
T_{b_1} \cap T_{b_2} = T_{b_1 1} \cap T_{b_2 1}, \quad b_1 \neq b_2 \tag{2.12}
$$

Cache Blocking for Gauss-Seidel

To implement cache blocking for Gauss-Seidel the key point to realize is that any elements in sub-block ℓ interact only with elements in sub-blocks $\ell - 1, \ell$, and $\ell + 1$. These are exactly the elements that Gauss-Seidel would require interacting through the edges of the level ℓ elements. Thus, for any particular block, update all blocks one time. Then, one can update the interior sub-blocks, reducing the domain that is updated for each sweep. Note that this method is consistent with regular Gauss-Seidel in that no elements are updated with neighboring unknowns which have not

[13] For the Pentium 4 being used, 512 KB cache exist. It is reasonable to assume that between 30% and 50% of L2 cache would be *owned* by our process on a lightly loaded machine.

[14] $N_c = N_s + 1$ where N_s is the number of repeated *sweeps* of the algorithm to be repeated.

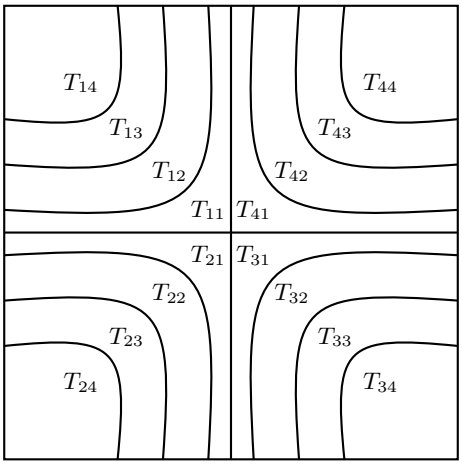


Figure 2.30: Block/SubBlock Partitioning for $\mathcal{T}$ with $N_b = 4, N_c = 4, N_s = 3$

been updated. Since the block was sized so that all necessary elements would fit in cache (L2), one will not be penalized with cache miss penalties. At this point, only the most interior elements in the block being updated have been updated the required number of times, i.e., the block is only partially updated.

One then continues through all of the remaining blocks and repeats the partial updating. At this point, one needs to *backtrack* through the blocks and finish the updating process for each block, proceeding from the boundary sub-block inward. The complete algorithm is presented in Algorithm 16.

For the situation depicted in Figure 2.30, the subblocks should be visited in the following order in order to be consistent with Gauss-Seidel sweeping through the complete domain three times:

- Primary Sweep:
  - $T_{14}, T_{13}, T_{12}, T_{11}, T_{14}, T_{13}, T_{12}, T_{14}, T_{13}$
  - $T_{24}, T_{23}, T_{22}, T_{21}, T_{24}, T_{23}, T_{22}, T_{24}, T_{23}$
  - $T_{34}, T_{33}, T_{32}, T_{31}, T_{34}, T_{33}, T_{32}, T_{34}, T_{33}$
  - $T_{44}, T_{43}, T_{42}, T_{41}, T_{44}, T_{43}, T_{42}, T_{44}, T_{43}$
- Backtracking Sweep:
  - T_{11}, T_{12}
  - T_{21}, T_{22}
  - T_{31}, T_{32}
  - T_{41}, T_{42}
  - T_{11}
  - T_{21}
  - T_{31}
  - T_{41}.

Algorithm 16 Smoother 1: `GS_Block_CB`

Require: Composite Mesh Level: $\mathcal{L}$, Composite Level Mesh : $\mathcal{T}_{\mathcal{L}}$
Require: Cache Blocking: $\mathcal{S}$, Number of Blocks: N_b, Number of sweeps: N_s
  Calculate ℓ_0 for global/local smoothing
  for $(b = 1; b \leq N_b; b\,{+}{+})$ **do**
    for $(i = 0; i < N_s; i\,{+}{+})$ **do**
      for $(c = N_s + 1; c > i; c\,{-}{-})$ **do**
        for $(\ell = \ell_0; \ell < \mathcal{L}; \ell\,{+}{+})$ **do**
          for all $K \in S^{\ell}_{bc}$ **do**
            $x_{K,\mathcal{L}} \leftarrow$ Updated over all DOF using point GS
          end for
        end for
        for all $K \in S^{\mathcal{L}}_{bc} \oplus S^{\mathcal{L},N}_{bc}$ **do**
          $x_{K,\mathcal{L}} \leftarrow$ Updated over all DOF using point GS
        end for
      end for
    end for
  end for
  for $(i = N_s - 1; i \geq 1; i\,{-}{-})$ **do**
    for $(b = 1; b \leq N_b; b\,{+}{+})$ **do**
      for $(c = 1; c \leq i; c\,{+}{+})$ **do**
        for $(\ell = \ell_0; \ell < \mathcal{L}; \ell\,{+}{+})$ **do**
          for all $K \in S^{\ell}_{bc}$ **do**
            $x_{K,\mathcal{L}} \leftarrow$ Updated over all DOF using point GS
          end for
        end for
        for all $K \in S^{\mathcal{L}}_{bc} \oplus S^{\mathcal{L},N}_{bc}$ **do**
          $x_{K,\mathcal{L}} \leftarrow$ Updated over all DOF using point GS
        end for
      end for
    end for
  end for

Figures 2.31–2.32 illustrates the effect of utilizing cache blocking coupled with Modified Block Sparse Row (MBSR) storage schemes. Note the following notation:

- NN_NN: No special performance optimizations,
- CB_NN: Cache Blocking only,
- NN_MB: MBSR storage,
- CB_MB: Cache Blocking and MBSR storage.

The following observations can be made:

- MBSR storage implies data contiguity of global stiffness matrix,
- Clear benefits in L2 cache misses using Cache Blocking,
- Clear benefits in TLB data misses using MBSR storage,
- Clear time savings using both.

2.10.3 Performance Monitoring

There are various performance monitoring tools available to the programmer. One option which is readily available is to turn on profiling via the `-pg` compiler option, and then process the resulting profile data with `gprof`. This will provide the programmer with either a function (or a line) profile of the time spent in each routine.

Charts presenting a comparison of the different optimization techniques as discussed above and applied to MG and PCG solvers for selected test cases are shown in Figures 2.33–2.36. Table 2.10 describes the notation of the different cases used in the figures. Note that the table describes only the last 4 characters, the meaning of `cg`, `mg`, `pcg` is clear.

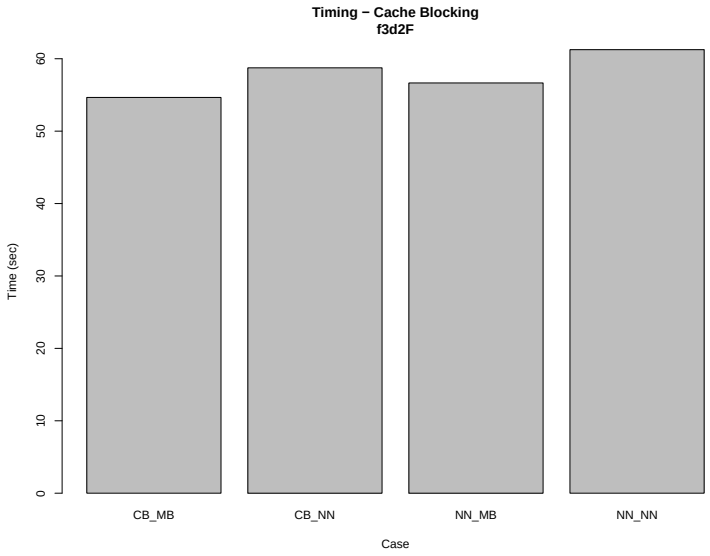
During the course of the development of the software for this research, much use was made of the `PAPI` performance counter monitoring software. `PAPI` provides the user with the ability to monitor the hardware performance counters which exist on most modern CPU architectures. The software is portable and functional on many different hardware architectures, although it should be pointed out that only a limited subset of the counters are available on the Intel Pentium 4 IA-32 architecture. The set of `PAPI` counters which could be accessed together during a run are listed in Table 2.11. For the f4, $r = 4$, adaptive run we illustrate a slightly different comparison utilizing `PAPI` performance monitoring variables as shown in Figures 2.37–2.40. For more on `PAPI`, see § 2.11.4.

Some observations need to be made here. All MFLOP/s rates presented are aggregate for a complete run, i.e., taking the total PAPI count of floating point operations divided by the total time. Only by looking at the accompanying detail graphs of the various cache misses does one obtain a clear indication (other than total solve time) as to effect of the different algorithms employed. It is clear that using MBSR storage coupled with cache blocking consistently seems to give the best results. Note also that the MBSR storage scheme coupled with cache blocking implies the overhead associated with creating the full MBSR matrix in storage. Even with this overhead, it is still competitive with non-MBSR schemes.

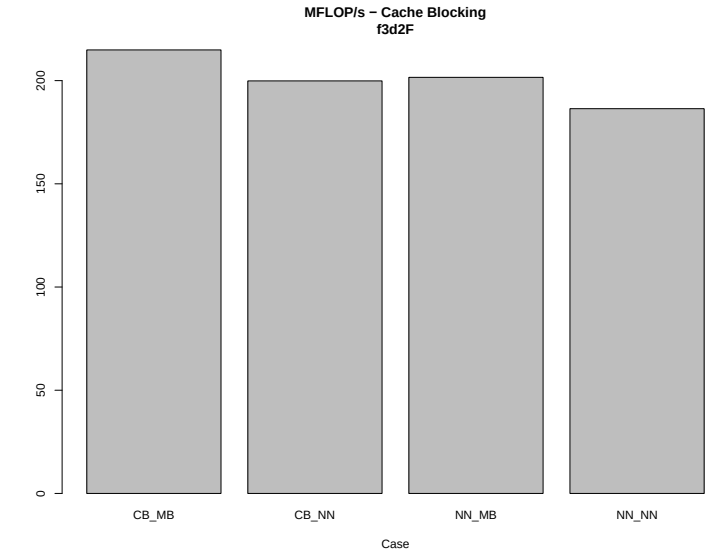
2.11 Auxiliary Software

2.11.1 `glib`

GTK+ and specifically `glib` provide an excellent API for use by programmers when developing applications to run in the UNIX environment. It is basically a standard developed for application

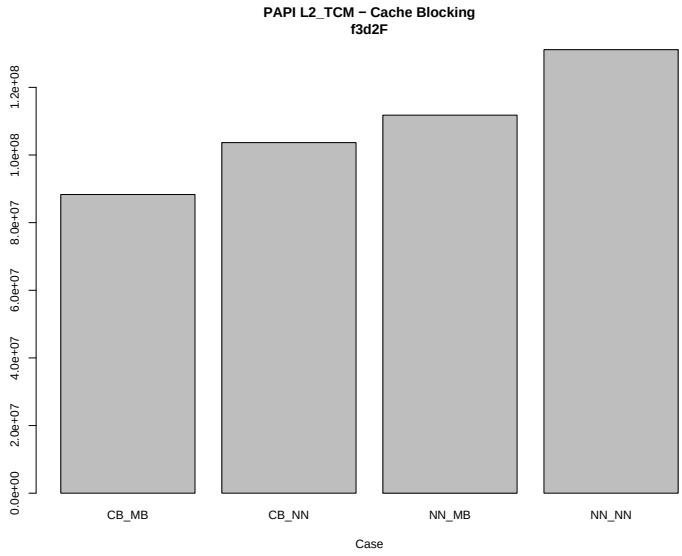


(a) Time (sec)

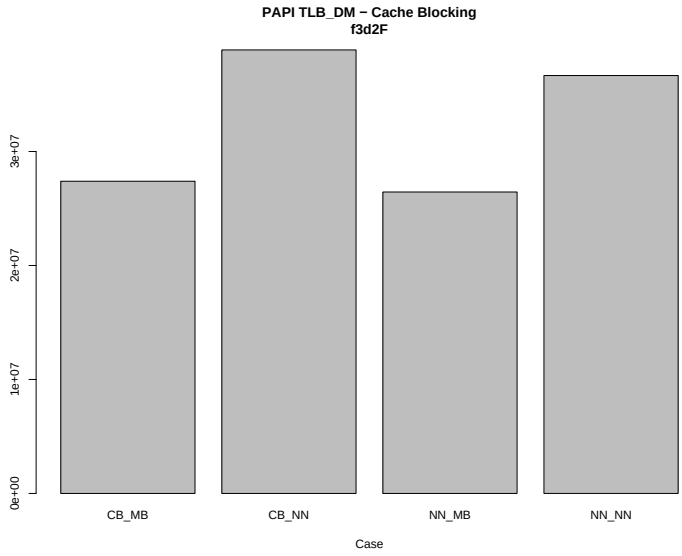


(b) MFLOP/s

Figure 2.31: Cache Blocking/MBSR Storage Comparison (Time): f3, $r = 3$, Uniform

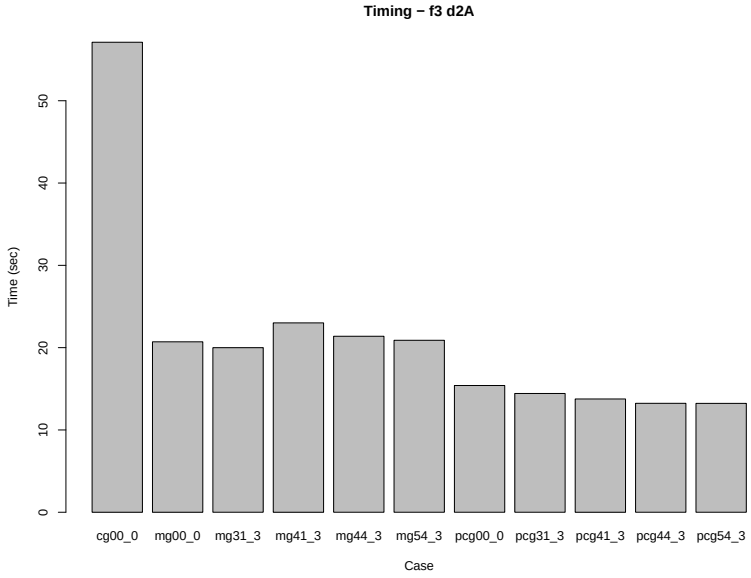


(a) PAPI_L2TCM

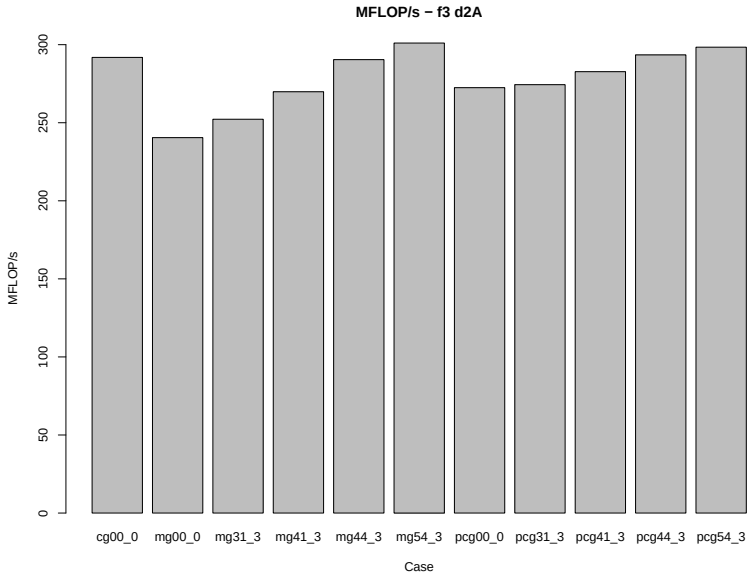


(b) PAPI_TLBDM

Figure 2.32: Cache Blocking/MBSR Storage Comparison (Cache Misses): f3, $r = 3$, Uniform

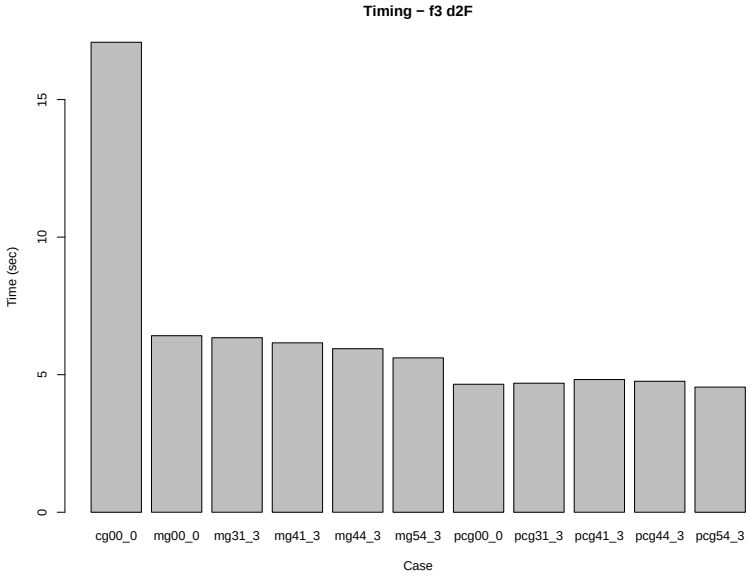


(a) Solver Time

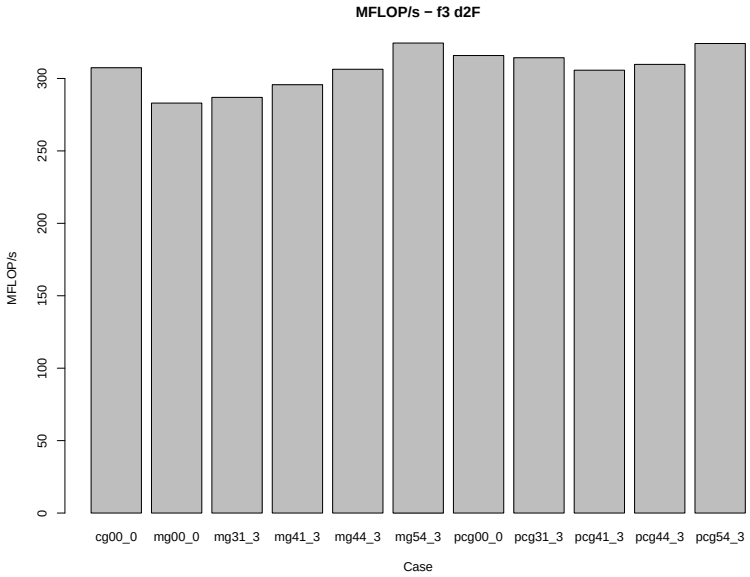


(b) Solver MFLOP/s

Figure 2.33: Solver Optimization Comparison : f3, $r = 3$, Adaptive

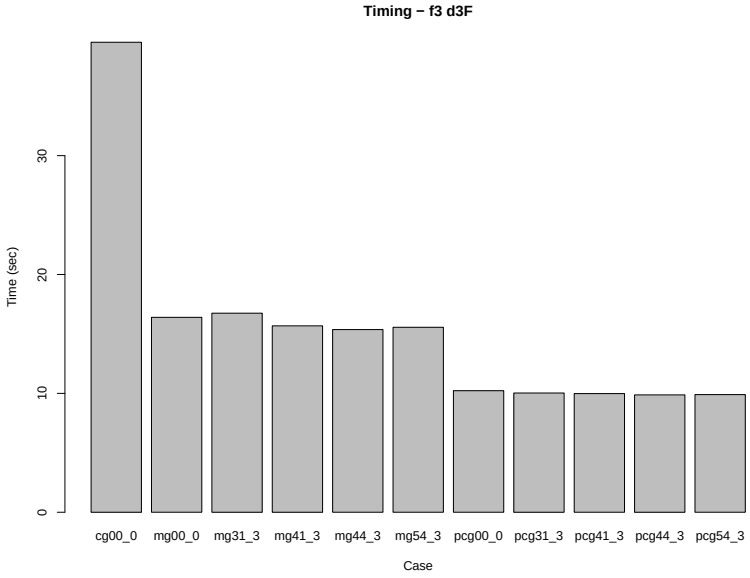


(a) Solver Time

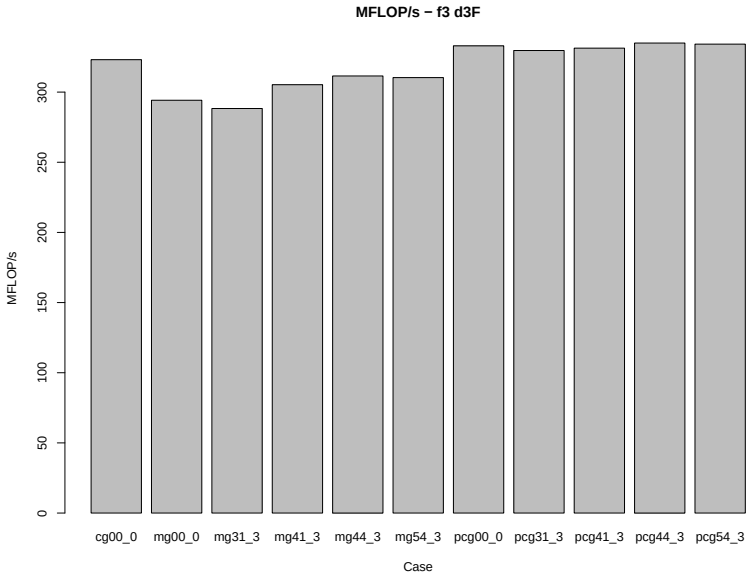


(b) Solver MFLOP/s

Figure 2.34: Solver Optimization Comparison : f3, $r = 3$, Uniform

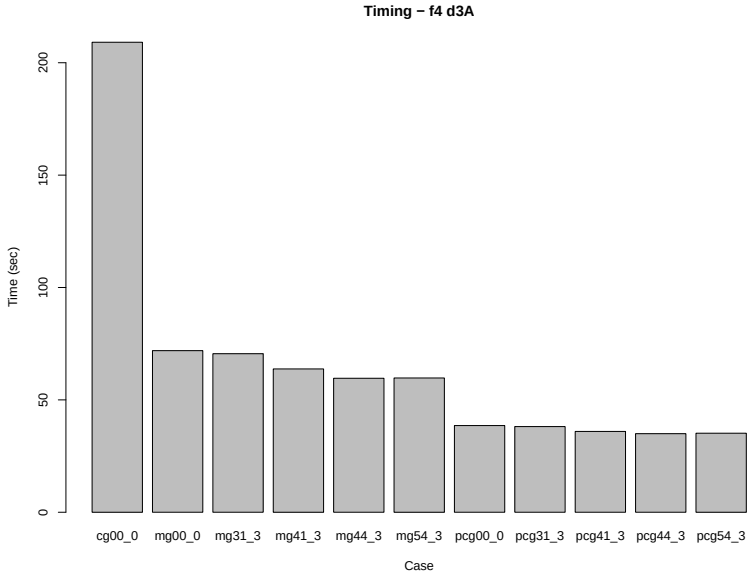


(a) Solver Time

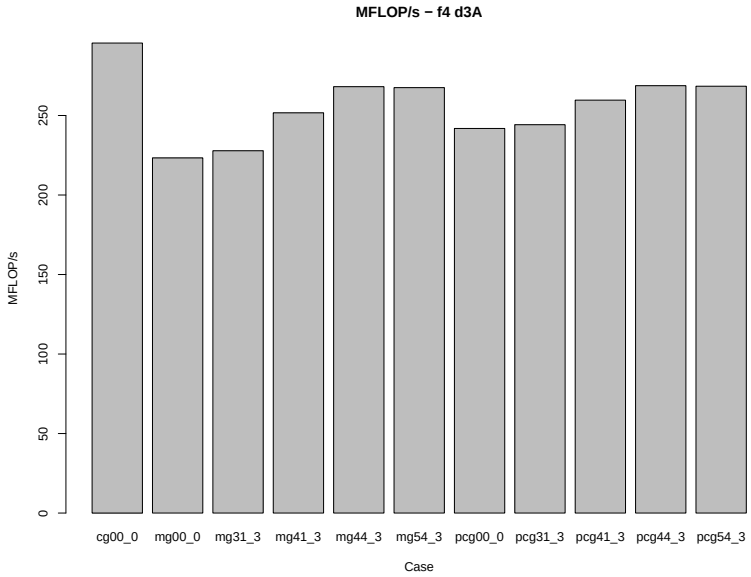


(b) Solver MFLOP/s

Figure 2.35: Solver Optimization Comparison : f3, $r = 4$, Uniform

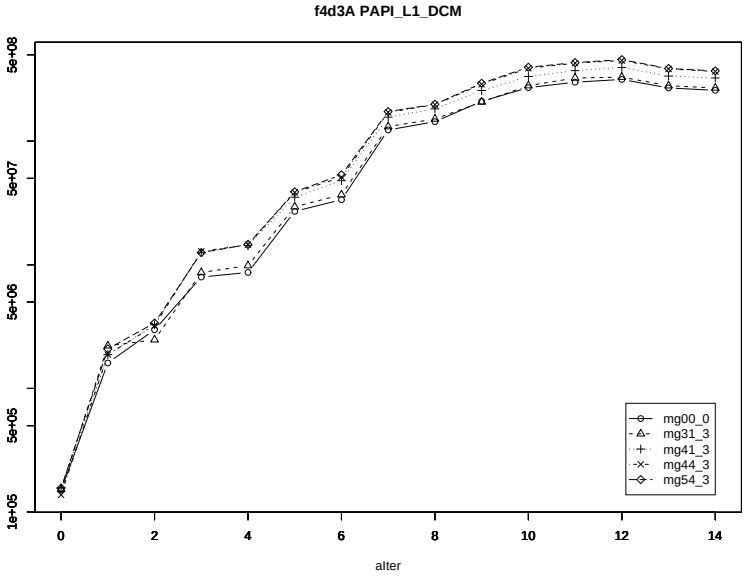


(a) Solver Time

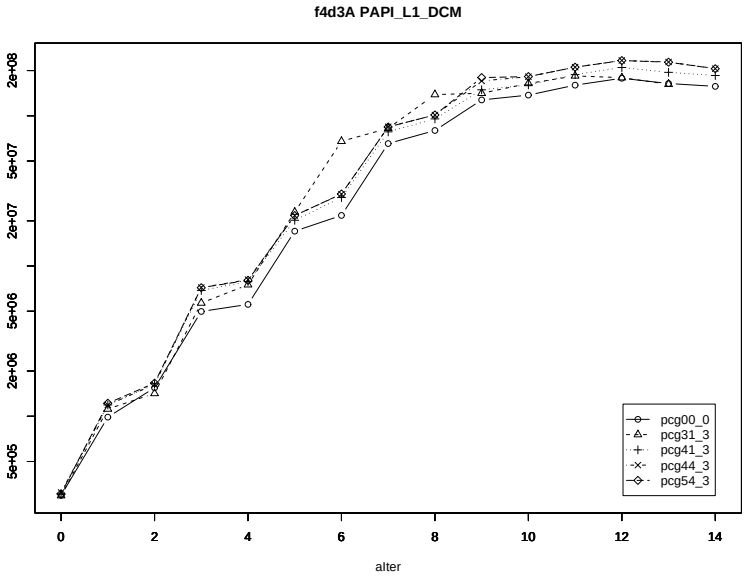


(b) Solver MFLOP/s

Figure 2.36: Solver Optimization Comparison : f4, $r = 4$, Adaptive

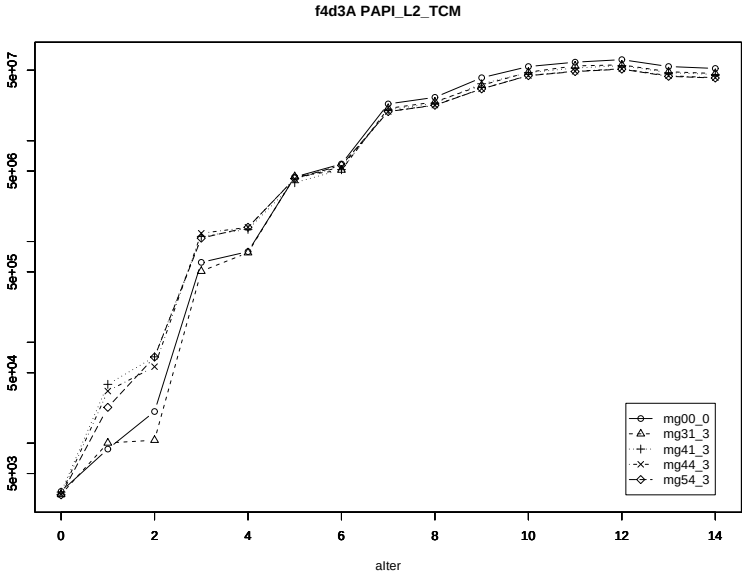


(a) MG

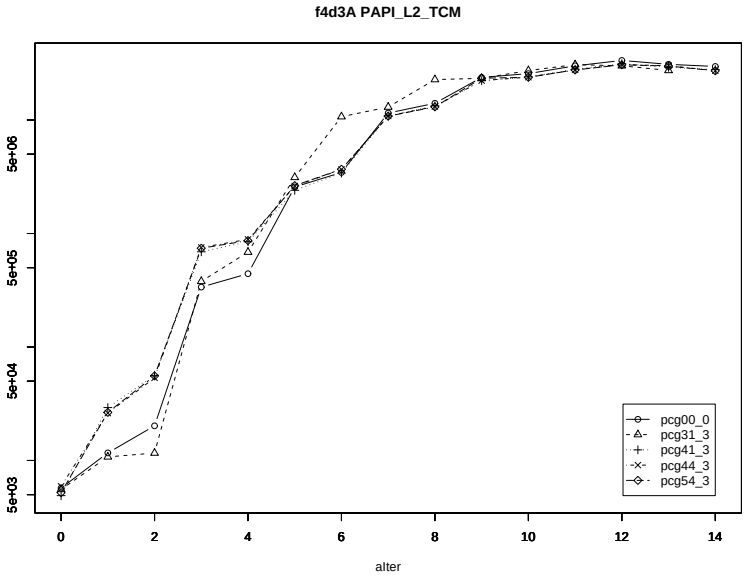


(b) PCG

Figure 2.37: PAPI_L1_DCM: f4, $r = 4$, Adaptive

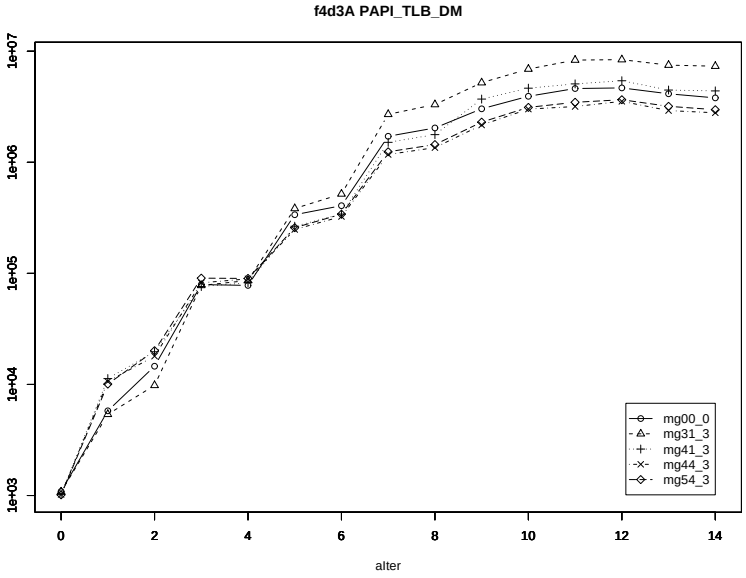


(a) MG

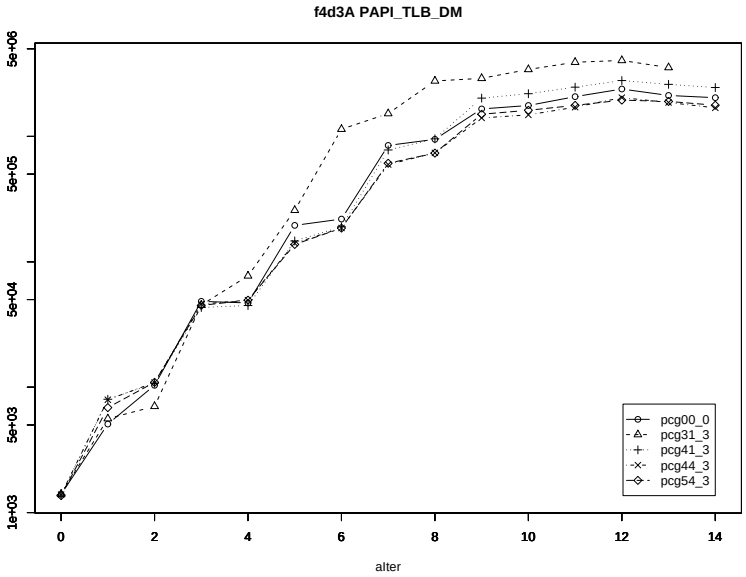


(b) PCG

Figure 2.38: PAPI_L2_TCM: f4, $r = 4$, Adaptive

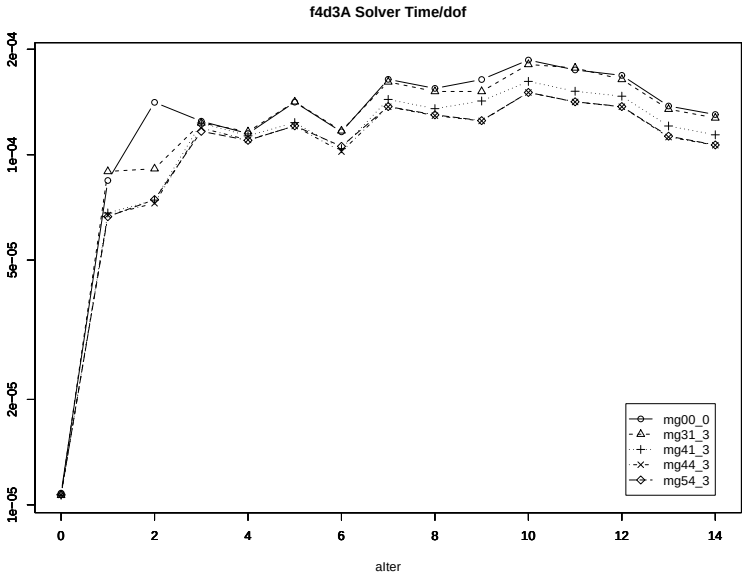


(a) MG

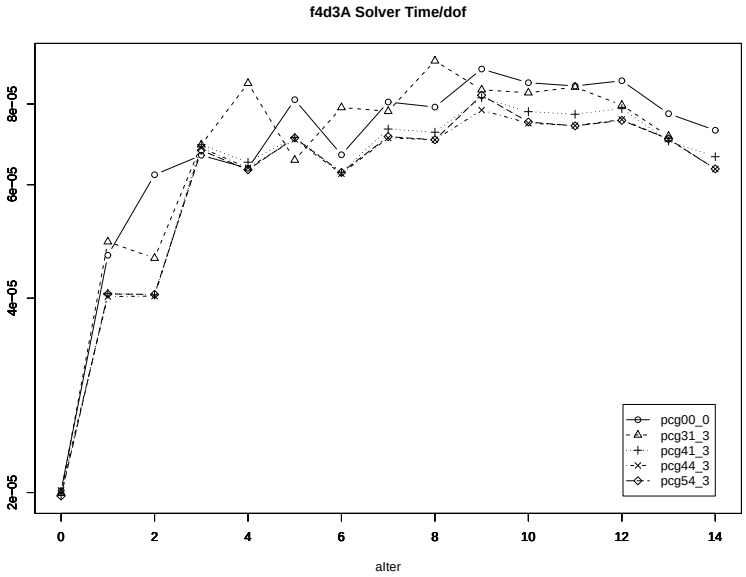


(b) PCG

Figure 2.39: `PAPI_TLB_DM`: f4, $r = 4$, Adaptive



(a) MG



(b) PCG

Figure 2.40: MFLOP/s: f4, $r = 4$, Adaptive

Table 2.10: Performance Case Key

Case	Description
00_0	No reordering of elements, no special GS
31_3	Reordering of Leaf elements, GS Cache Blocking on Leaf
41_3	Reordering of Leaf elements - MBSR routines used on Leaf
44_3	Reordering of Leaf elements - MBSR routines used on Leaf and NonLeaf
54_3	Reordering of Leaf elements - MBSR routines used on Leaf and NonLeaf, GS Cache Blocking on Leaf

Table 2.11: `PAPI` Hardware Counters

Counter	Description
`PAPI_TOT_CYC`	CPU cycles
`PAPI_FP_OPS`	Floating Point Operations retired
`PAPI_TOT_IIS`	CPU instructions issued
`PAPI_TOT_INS`	CPU instructions retired
`PAPI_RES_STL`	CPU cycles resources stalled
`PAPI_L1_DCM`	L1 Data Cache misses
`PAPI_L2_TCM`	L2 Total Cache misses
`PAPI_TLB_DM`	Translation Lookaside Buffer (TLB) Data misses

development using the GNOME framework. Figures 2.41–2.43 illustrate standard glib objects used throughout this thesis and referenced in Sections 2.3 and 2.5. For more information regarding this library, see http://www.gtk.org/ and http://developer.gnome.org/arch/gtk/glib.html.

2.11.2 triangle

Triangle is a two-dimensional quality mesh generator and Delaunay triangulator written by Jonathan Richard Shewchuk currently at the University of California at Berkeley. It allows one to input the domain in a very easy fashion as a planar straight line graph. I link this package in to produce all initial meshes. It is a quick, reliable, and quite flexible mesh generator which produces quality quasiuniform meshes. For more information, see Shewchuk (1996) and http://www.cs.cmu.edu/~quake/triangle.html.

2.11.3 ATLAS

ATLAS provides optimized versions of BLAS, which can be used seamlessly with clapack routines. These routines are used throughout the implementation of DG-FEM wherever BLAS operations can be used, such as matrix-vector operations for projection and embedding and the coarse level solve for multilevel solvers. For more information on ATLAS, see Whaley et al. (2001) and http://math-atlas.sourceforge.net/.

2.11.4 PAPI

Quoting the PAPI web page:

> PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. PAPI enables software engineers to see, in near real time, the relation between software performance and processor events.

Note that during the course of use of PAPI, when code was compiled with SSE2 extensions, i.e., vector operations on multiple doubles at one time, PAPI had some issues with producing accurate floating point operation counts. The numbers here do not reflect attempts to have the compiler optimize for vector operations and thus I feel they are representative of actual floating point counts. For more information on PAPI, see Browne et al. (2000) and http://icl.cs.utk.edu/papi/.

2.11.5 R

Quoting from the R web page:

> R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.
>
> R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

```
typedef struct _GList GList;
struct _GList
{
  gpointer data;
  GList *next;
  GList *prev;
};
```

Figure 2.41: GList Object Definition

```
typedef struct _GSList GSList;
struct _GSList
{
  gpointer data;
  GSList *next;
};
```

Figure 2.42: GSList Object Definition

```
typedef struct _GNode GNode;
struct _GNode
{
  gpointer data;
  GNode *next;
  GNode *prev;
  GNode *parent;
  GNode *children;
};
```

Figure 2.43: GNode Object Definition

All of the charts and graphs included in this dissertation were generated with R. I highly recommend obtaining and learning to use R for anyone who has to process large amounts of data. For more information on R, see `http://www.r-project.org/`.

2.11.6 `meshtv/silo`

Quoting from the Meshtv web page:

> MeshTV is an interactive graphical analysis tool for visualizing and analyzing data on two- and three-dimensional (2D, 3D) meshes. It is a general purpose tool that handles many different mesh types, provides different ways of viewing the data, and is virtually hardware/vendor independent while still providing graphics at the speed of the native graphics hardware.

MeshTV utilizes the SILO database format, which implements an application program interface expressly designed for accessing scientific data. The SILO library defines a set of objects for handling different meshes and the SILO API can also be used outside of visualization for checkpoint and restart file management. For more information on MeshTV and SILO, see `http://www.llnl.gov/bdiv/meshtv/`.

2.11.7 `METIS`

Quoting from the METIS web page,

> METIS is a family of programs for partitioning unstructured graphs and hypergraphs and computing fill-reducing orderings of sparse matrices. The underlying algorithms used by METIS are based on the state-of-the-art multilevel paradigm that has been shown to produce high quality results and scale to very large problems.

METIS is used to partition the domain into blocks for use in the Douglas cache blocking scheme. Note that since METIS is already being used, migrating our implementation to a parallel version on large clusters should be fairly straightforward. For more information on METIS, see Karypis and Kumar (1998), Karypis and Kumar (1995) and `http://glaros.dtc.umn.edu/gkhome/views/metis`.

2.11.8 Miscellaneous

There were many other software packages which aided in the development and assembly of this dissertation. Since code development was done on UNIX based machines, `Perl` was utilized to filter through large datasets and aggregate summary data. In addition the `gcc` compiler systems coupled with the `gprof` utility was useful in identifying bottlenecks by profiling time spent in each routine. `Maple` was instrumental in validating derivation of basis functions and test problems.

Chapter 3

Second Order Elliptic Problems

3.1 DG Formulation

3.1.1 Model Problem

The second order elliptic model problem under consideration is:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = g_D & \text{on } \Gamma_D \\ \nabla u \cdot n = g_N & \text{on } \Gamma_N \end{cases} \tag{3.1}$$

where $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ and $\partial\Omega = \Gamma = \Gamma_D \cup \Gamma_N$ with n being the unit outward normal vector to Γ.

Let $\mathcal{T}_h = \{K_i : i = 1, 2, \ldots, m_h\}$ be a family of star-like partitions of Ω parameterized by $0 < h \leq 1$. The elements of $\mathcal{T}_h$ satisfy the minimal angle condition and $\mathcal{T}_h$ is locally quasi-uniform. Let the edges satisfy $\mathcal{E}^I = \{e = \partial K_j \cap \partial K_l : \mu_{d-1}(\partial K_j \cap \partial K_l) > 0\}$, $\mathcal{E}^B = \{e = \partial K_j \cap \partial\Omega : \mu_{d-1}(\partial K_j \cap \partial\Omega) > 0\}$. Note that $\forall e \in \mathcal{E}^B$, either $e \subset \Gamma_D$ or $e \subset \Gamma_N$ and $\mathcal{E} = \mathcal{E}^I \cup \mathcal{E}^B$, where $\mathcal{E}^B = \mathcal{E}_D^B \cup \mathcal{E}_N^B$ and $\mathcal{E}_D^B \cap \mathcal{E}_N^B = \emptyset$. Also note that if $e \in \mathcal{E}^I$, then $e = \partial K^+ \cap \partial K^-$ for $K^+, K^- \in \mathcal{T}_h$. Similarly if $e \in \mathcal{E}^B$, then $e = \partial K^+ \cap \partial\Omega \equiv \partial K \cap \partial\Omega$. Finally, n^+ is the unit normal to e that points outward from K^+. Let the *energy spaces* E_h be defined as

$$E_h = \prod_{K \in \mathcal{T}_h} H^2(K)$$

and the *finite element spaces* V_h^r be defined as

$$V_h^r = \prod_{K \in \mathcal{T}_h} P_r(K)$$

where $P_r(K)$ denotes the space of polynomials of total degree $r - 1$. Note that $V_h^r \subset E_h \subset L^2(\Omega)$, but $V_h^r \not\subset H^1(\Omega)$.

3.1.2 The DG Method

First obtain weak formulation by multiplying Eq 3.1 by $v \in V_h^r$ and integrating over Ω:

$$-\int_\Omega (\Delta u)v \; dx = \int_\Omega fv \; dx.$$

Next decompose integrals into element contributions and integrate by parts:

$$\sum_{K\in\mathcal{T}_h} -\int_K (\Delta u)v \; dx = \sum_{K\in\mathcal{T}_h} \int_K fv \; dx$$
$$\sum_{K\in\mathcal{T}_h} \int_K \nabla u \cdot \nabla v \; dx - \sum_{K\in\mathcal{T}_h} \int_{\partial K} \frac{\partial u}{\partial n} v \; ds = \sum_{K\in\mathcal{T}_h} \int_K fv \; dx$$

The next step is to split edge integrals:

$$\sum_{K\in\mathcal{T}_h} \left\langle \frac{\partial u}{\partial n}, v \right\rangle_{\partial K} = \sum_{e\in\Gamma_D} \left\langle \frac{\partial u}{\partial n}, v \right\rangle_e + \sum_{e\in\Gamma_N} \left\langle \frac{\partial u}{\partial n}, v \right\rangle_e + \sum_{e\in\mathcal{E}^I} \left(\left\langle \frac{\partial u^+}{\partial n^+}, v^+ \right\rangle_e + \left\langle \frac{\partial u^-}{\partial n^-}, v^- \right\rangle_e \right)$$

resulting in:

$$\sum_{K\in\mathcal{T}_h} (\nabla u, \nabla v)_K - \left\langle \frac{\partial u}{\partial n}, v \right\rangle_{\Gamma_D} - \sum_{e\in\mathcal{E}^I} \left(\left\langle \frac{\partial u^+}{\partial n^+}, v^+ \right\rangle_e - \left\langle \frac{\partial u^-}{\partial n^+}, v^- \right\rangle_e \right) = \sum_{K\in\mathcal{T}_h} (f, v)_K + \langle g_N, v \rangle_{\Gamma_N}$$

since g_N is given.

There are several different ways of working with above internal edge integrals, here we present two:

- D. Arnold (Arnold, 1982):

$$ac - bd = \frac{1}{2}(a+b)(c-d) + \frac{1}{2}(a-b)(c+d), \tag{3.2}$$

  implies

$$\begin{aligned} \frac{\partial u^+}{\partial n^+}v^+ - \frac{\partial u^-}{\partial n^+}v^- &= \frac{1}{2}\left(\frac{\partial u^+}{\partial n^+} + \frac{\partial u^-}{\partial n^+}\right)\left(v^+ - v^-\right) + \frac{1}{2}\left(\frac{\partial u^+}{\partial n^+} - \frac{\partial u^-}{\partial n^+}\right)\left(v^+ + v^-\right) \\ &= \left\{\frac{\partial u}{\partial n}\right\}[v] + \{v\}\left[\frac{\partial u}{\partial n}\right] \\ &= \{\partial_n u\}[v] + \{v\}[\partial_n u] \end{aligned} \tag{3.3}$$

  where $\partial_n v := \dfrac{\partial v}{\partial n} = n \cdot \nabla v$, $\{\partial_n u\}\big|_e = \dfrac{1}{2}\left(\partial_n u^+ + \partial_n u^-\right)\big|_e$, and $[v]\big|_e = \left(v^+ - v^-\right)\big|_e$.

- G. Baker (Baker, 1977):

$$ac - bd = a(c-d) + (a-b)d. \tag{3.4}$$

implies

$$\begin{aligned}\frac{\partial u^+}{\partial n^+}v^+ - \frac{\partial u^-}{\partial n^+}v^- &= \frac{\partial u^+}{\partial n^+}[v] + \left[\frac{\partial u}{\partial n}\right]v^- = \frac{\partial u^+}{\partial n}[v] + \left[\frac{\partial u}{\partial n}\right]v^- \\ &= \left\{\frac{\partial u}{\partial n}\right\}[v] + \left[\frac{\partial u}{\partial n}\right]v^- \\ &= \{\partial_n u\}[v] + [\partial_n u]\,v^-\end{aligned} \tag{3.5}$$

where$\{\partial_n u\}\big|_e = \partial_n u^+\big|_e$.

Note that for $u \in H^2(\Omega)$, fluxes are continuous across interelement boundaries and

$$\int_{e\in\mathcal{E}^I}\left(\{\partial_n u\}[v] + \{v\}[\partial_n u]\right)\,ds = \int_{e\in\mathcal{E}^I}\{\partial_n u\}[v]\,ds \qquad (Arnold)$$

$$\int_{e\in\mathcal{E}^I}\left(\{\partial_n u\}[v] + [\partial_n u]\,v^-\right)\,ds = \int_{e\in\mathcal{E}^I}\{\partial_n u\}[v]\,ds \qquad (Baker).$$

Define

- $B(u,v) := \sum_{K\in\mathcal{T}_h}(\nabla u, \nabla v)_K$
- $F(v) := \sum_{K\in\mathcal{T}_h}(f,v)_K + \langle g_N, v\rangle_{\Gamma_N}$
- $J(u,v) := \langle \partial_n u, v\rangle_{\Gamma_D} + \sum_{e\in\mathcal{E}^I}\langle\{\partial_n u\},[v]\rangle_e$

This leads to the weak formulation of Eq 3.1: Find $u \in H^2(\Omega)$ such that

$$B(u,v) - J(u,v) = F(v) \quad \forall v \in E_h \tag{3.6}$$

3.1.3 SIPG Formulation

One can now make some modifications in order to provide the bilinear form with certain desirable properties, symmetry and coercivity. Noting $J(v,u) = 0$ for smooth u implies that for symmetry,

$$B(u,v) - J(u,v) - J(v,u) = F(v) - \langle \partial_n v, g_D\rangle_{\Gamma_D}.$$

For coercivity, one can penalize the jump terms. First define a penalization parameter $\sigma = \sigma(\gamma, h_e, r) > 0$. Again noting that integrals involving a jump in u are zero for smooth u and that g_D is given data, then

$$J^\sigma(u,v) := \sum_{e\in\mathcal{E}^I}\langle\sigma[u],[v]\rangle_e + \langle\sigma u, v\rangle_{\Gamma_D}$$

which leads to

Problem (SIPG Formulation). *Find $u \in H^1 \cap E_h$ such that*

$$B(u,v) - J(u,v) - J(v,u) + J^\sigma(u,v) = F(v) - \langle\partial_n v, g_D\rangle_{\Gamma_D} + \langle\sigma g_D, v\rangle_{\Gamma_D} \quad \forall v \in E_h. \tag{3.7}$$

Thus

Problem (DG Formulation). *Find* $u_h^\gamma \in V_h^r$ *such that*

$$a_h^\gamma\left(u_h^\gamma, v\right) = F_h^\gamma(v), \qquad \forall v \in V_h^r \tag{3.8}$$

where

$$a_h^\gamma\left(u_h^\gamma, v\right) = \sum_{K\in\mathcal{T}_h} \left(\nabla u_h^\gamma, \nabla v\right)_K - \sum_{e\in\mathcal{E}^I\cup\mathcal{E}_D^B} \left(\left\langle \left\{\partial_n u_h^\gamma\right\}, [v] \right\rangle_e + \left\langle \left\{\partial_n v\right\}, [u_h^\gamma] \right\rangle_e - \gamma h_e^{-1} \left\langle [u_h^\gamma], [v] \right\rangle_e \right) \tag{3.9}$$

and

$$F_h^\gamma(v) = \sum_{K\in\mathcal{T}_h} (f, v)_K - \sum_{e\in\Gamma_D} \left\langle g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{e\in\Gamma_N} \left\langle g_N, v \right\rangle_e . \tag{3.10}$$

The bilinear form $a_h^\gamma(\cdot,\cdot)$ induces the following norm on E_h:

$$\|v\|_{1,h} = \left(\sum_{K\in\mathcal{T}_h} \|\nabla v\|_{0,K}^2 + \sum_{e\in\mathcal{E}^I\cup\mathcal{E}_D^B} \left(h_e^{-1} |[v]|_{0,e}^2 + h_e \left|\{\partial_n v\}\right|_{0,e}^2 \right) \right)^{1/2} \tag{3.11}$$

Recall that we have been using two different forms for $\{\partial_n v\}\big|_e$:

1. **Arnold**: $\{\partial_n v\}\big|_e = \dfrac{1}{2}\left(\dfrac{\partial v^+}{\partial n^+} + \dfrac{\partial v^-}{\partial n^+} \right)\Bigg|_e$
2. **Baker**: $\{\partial_n v\}\big|_e = \dfrac{\partial v^+}{\partial n^+}\Bigg|_e$

and of course $[v]\big|_e = v^+\big|_e - v^-\big|_e$.

3.1.4 Stiffness Matrix Assembly

Assembly of the stiffness matrix A requires subassembly of the diagonal blocks A_{ii} and the off-diagonal blocks A_{ij}, $i,j = 1,2,\ldots,|\mathcal{T}_h|$. Note that $A_{ii} = A_{ii}^\mathsf{T}$ and $A_{ij} = A_{ji}^\mathsf{T}$ implies that $A = A^\mathsf{T}$. Thus, assembly of A requires subassembly of $A_K, \forall K \in \mathcal{T}_h$ and $A_e, \forall e \in \mathcal{E}^I$. Note here that since each interior edge e is the common boundary between two triangles K^+, K^-, A_e is the off-diagonal block describing interaction between degrees of freedom of K^+ and degrees of freedom of K^- through edge e.

The first step is to rewrite Eq (3.9) into a form where individual components can be clearly identified as being associated with either A_K or A_e. Expanding the jump terms gives:

$$\begin{aligned} a_h^\gamma(u,v) = &\sum_{K\in\mathcal{T}_h} (\nabla u, \nabla v)_K - \sum_{e\in\mathcal{E}^I\cup\mathcal{E}_D^B} \left(\left\langle \{\partial_n u\}, v^+ \right\rangle_e + \left\langle \{\partial_n v\}, u^+ \right\rangle_e - \gamma h_e^{-1} \left\langle u^+, v^+ \right\rangle_e \right) \\ &+ \sum_{e\in\mathcal{E}^I} \left(\left\langle \{\partial_n u\}, v^- \right\rangle_e + \left\langle \{\partial_n v\}, u^- \right\rangle_e + \gamma h_e^{-1} \left\langle u^-, v^- \right\rangle_e - \gamma h_e^{-1} \left\langle u^+, v^- \right\rangle_e - \gamma h_e^{-1} \left\langle u^-, v^+ \right\rangle_e \right) \end{aligned} \tag{3.12}$$

A couple of notes regarding Eq (3.12). First, any terms involving both u^+, v^+ or u^-, v^- are part of A_{K^+} or A_{K^-}, respectively. Second, any terms involving u^+, v^- or u^-, v^+ are a part of A_e. Third,

Eq (3.12) takes different forms depending on whether the Arnold or Baker forms of $\{\partial_n u\}$ and $\{\partial_n v\}$ are chosen. For the Baker formulation, Eq (3.12) becomes

$$a_h^\gamma(u,v) = \sum_{K\in\mathcal{T}_h} (\nabla u, \nabla v)_K - \sum_{e\in\mathcal{E}^I\cup\mathcal{E}_D^B} \Big(\langle \partial_n u^+, v^+ \rangle_e + \langle \partial_n v^+, u^+ \rangle_e - \gamma h_e^{-1} \langle u^+, v^+ \rangle_e \Big)$$
$$+ \sum_{e\in\mathcal{E}^I} \Big(\langle \partial_n u^+, v^- \rangle_e + \langle \partial_n v^+, u^- \rangle_e - \gamma h_e^{-1} \langle u^+, v^- \rangle_e - \gamma h_e^{-1} \langle u^-, v^+ \rangle_e \Big) + \sum_{e\in\mathcal{E}^I} \gamma h_e^{-1} \langle u^-, v^- \rangle_e \tag{3.13}$$

and for the Arnold formulation

$$a_h^\gamma(u,v) = \sum_{K\in\mathcal{T}_h} (\nabla u, \nabla v)_K - \sum_{e\in\mathcal{E}^I\cup\mathcal{E}_D^B} \Big(\frac{1}{2}\langle \partial_n u^+, v^+ \rangle_e + \frac{1}{2}\langle \partial_n v^+, u^+ \rangle_e - \gamma h_e^{-1} \langle u^+, v^+ \rangle_e \Big)$$
$$+ \sum_{e\in\mathcal{E}^I} \Big(\frac{1}{2}\langle \partial_n u^+, v^- \rangle_e + \frac{1}{2}\langle \partial_n v^+, u^- \rangle_e - \frac{1}{2}\langle \partial_n u^-, v^+ \rangle_e - \frac{1}{2}\langle \partial_n v^-, u^+ \rangle_e - \gamma h_e^{-1} \langle u^+, v^- \rangle_e$$
$$- \gamma h_e^{-1} \langle u^-, v^+ \rangle_e \Big) + \sum_{e\in\mathcal{E}^I} \Big(\frac{1}{2}\langle \partial_n u^-, v^- \rangle_e + \frac{1}{2}\langle \partial_n v^-, u^- \rangle_e + \gamma h_e^{-1} \langle u^-, v^- \rangle_e \Big) \tag{3.14}$$

3.2 A Posteriori Error Estimation

A posteriori error estimators are used to determine regions where refinement and coarsening should be done under an adaptive FEM framework. The residual type a posteriori estimator is the same as the one described in Karakashian and Pascal (2004, 2006, Theorem 3.1). The local problem type a posteriori estimator is the same as the one the one described in Karakashian and Pascal (2003, §4.1), including explicit calculation of the local problem right hand side for both the Arnold and Baker SIPG formulation.

3.2.1 Residual A Posteriori Estimator

The following residual type a posteriori estimator theorem is used as our residual estimator and is stated without proof.

Theorem 3.2.1 (Karakashian and Pascal (2004, 2006, Theorem 3.1)). *Let $e = u - u_h^\gamma$. Then*

$$\sum_{K\in\mathcal{T}_h} \|\nabla e\|_K^2 \le c\Big(\sum_{K\in\mathcal{T}_h} h_K^2 \|f + \Delta u_h^\gamma\|_K^2 + \sum_{e\in\mathcal{E}^I} h_e |[\partial_n u_h^\gamma]|_e^2 + \sum_{e\in\mathcal{E}_N^B} h_e |g_N - \partial_n u_h^\gamma|_e^2$$
$$+ \gamma^2 \sum_{e\in\mathcal{E}^I} h_e^{-1} |[u_h^\gamma]|_e^2 + \gamma^2 \sum_{e\in\mathcal{E}_D^B} h_e^{-1} |g_D - u_h^\gamma|_e^2 \Big).$$

Note that the constant c is a constant depending only on r and the minimum angle θ of the triangles in the initial triangulation.

3.2.2 Local A Posteriori Estimator

The approach here follows that of Karakashian and Pascal (2003, §$\tilde{4}$.1) and is based on domain decomposition ideas contained in Feng and Karakashian (2001). Basically we view the computed solution u_h^γ as a "coarse-mesh" approximation to some function in a higher dimensional subspace which is arguably a more accurate approximation to the solution u. We do not compute this more accurate approximation directly, but attempt to approximate it by adding to u_h^γ a function computed as the solution of "local" problems. These local problems are formulated in a higher dimensional space on each triangle $K \in \mathcal{T}_h$ obtained through either h or p refinement.

Define the finite element space $V' := V_h^{r'}$ consisting of discontinuous piecewise polynomial functions of degree less than or equal to $r'-1$ where $r' \geq r$. Note that V_h^r is a subspace of V'. On $V' \times V'$ we define the bilinear form $a' := a_{h'}^{\gamma'}$, where a_h^γ is the restriction of a' to V_h^r in the sense that

$$a_h^\gamma(v,w) - a'(v,w) \qquad \forall v, w \in V_h^r. \tag{3.15}$$

For each $K \in \mathcal{T}_h$ we consider the "local" space $V'(K)$ obtained by restricting V' to K. By extending the elements of $V'(K)$ by zero to the rest of Ω, $V'(K)$ becomes a subspace of V'. On $V'(K) \times V'(K)$ introduce the bilinear form $a'_K(\cdot,\cdot)$ as the restriction of $a'(\cdot,\cdot)$ to $V'(K) \times V'(K)$. Note that a'_K inherits the symmetry and coercivity of a' on $V'(K)$.

Now let $u' := u_{h'}^{\gamma'}$ be the discontinuous Galerkin approximation of u in the space V', i.e.,

$$a'(u',v) = (f,v) - \sum_{e \in \Gamma_D} \left\langle g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{e \in \Gamma_N} \langle g_N, v \rangle_e \qquad \forall v \in V'. \tag{3.16}$$

Noting the orthogonality relation:

$$a'(u' - u_h^\gamma, v) = 0 \qquad \forall v \in V_h^r$$

allows us to pose the local problems to be solved for the functions $\{\eta_K \in V'(K) | K \in \mathcal{T}_h\}$ as:

$$a'_K(\eta_K, v) = (f,v) - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \left\langle g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle g_N, v \rangle_e - a'(u_h^\gamma, v) \qquad \forall v \in V'(K). \tag{3.17}$$

Before getting started, note the integration by parts formula for a single element K

$$(\nabla u, \nabla v)_K = (-\Delta u, v)_K + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle \partial_n u, v \rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \langle \partial_n u, v \rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^+}} \left\langle \partial_n u^+, v^+ \right\rangle_e - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^-}} \left\langle \partial_n u^-, v^- \right\rangle_e. \tag{3.18}$$

Then

$$
\begin{aligned}
a'_K(\eta_K, v) &= (f, v)_K - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \left\langle g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle g_N, v \rangle_e - a'(u_h^\gamma, v) \\
&= (f, v)_K - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \left\langle g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle g_N, v \rangle_e \\
&\quad - \left[(\nabla u_h^\gamma, \nabla v)_K - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \cup \mathcal{E}_D^B}} \left(\langle \{\partial_n u_h^\gamma\}, [v] \rangle_e + \langle \{\partial_n v\}, [u_h^\gamma] \rangle_e - \gamma h_e^{-1} \langle [u_h^\gamma], [v] \rangle_e \right) \right]
\end{aligned} \tag{3.19}
$$

Substituting Eq 3.18 into Eq 3.19 gives

$$
\begin{aligned}
a'_K(\eta_K, v) = (f + \Delta u_h^\gamma, v)_K &+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \left\langle u_h^\gamma - g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle g_N - \partial_n u_h^\gamma, v \rangle_e \\
&+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I}} \left(\langle \{\partial_n u_h^\gamma\}, [v] \rangle_e + \left\langle [u_h^\gamma], \{\partial_n v\} - \gamma h_e^{-1} [v] \right\rangle_e \right) \\
&- \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^+}} \left\langle \partial_n u^+, v^+ \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^-}} \left\langle \partial_n u^-, v^- \right\rangle_e .
\end{aligned} \tag{3.20}
$$

For the Baker formulation, Eq 3.20 becomes

$$
\begin{aligned}
a'_K(\eta_K, v) &= (f + \Delta u_h^\gamma, v)_K + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \left\langle u_h^\gamma - g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle g_N - \partial_n u_h^\gamma, v \rangle_e \\
&\quad + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I}} \left(\left\langle \partial_n \left(u_h^\gamma\right)^+, [v] \right\rangle_e + \left\langle [u_h^\gamma], \partial_n v^+ - \gamma h_e^{-1} [v] \right\rangle_e \right) \\
&\quad - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^+}} \left\langle \partial_n u^+, v^+ \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^-}} \left\langle \partial_n u^-, v^- \right\rangle_e \\
&= (f + \Delta u_h^\gamma, v)_K + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_D^B}} \left\langle u_h^\gamma - g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}_N^B}} \langle g_N - \partial_n u_h^\gamma, v \rangle_e \\
&\quad + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^+}} \left\langle [u_h^\gamma], \partial_n v^+ - \gamma h_e^{-1} v^+ \right\rangle_e + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^-}} \left(\left\langle [u_h^\gamma], \gamma h_e^{-1} v^- \right\rangle_e - \left\langle [\partial_n u_h^\gamma], v^- \right\rangle_e \right).
\end{aligned} \tag{3.21}
$$

For the Arnold formulation, Eq 3.20 becomes

$$
\begin{aligned}
a'_K(\eta_K, v) &= (f + \Delta u_h^\gamma, v)_K + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}_D^B}} \left\langle u_h^\gamma - g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}_N^B}} \left\langle g_N - \partial_n u_h^\gamma, v \right\rangle_e \\
&\quad + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I}} \left(\left\langle \frac{1}{2}\left((u_h^\gamma)^+ + (u_h^\gamma)^-\right), [v] \right\rangle_e + \left\langle [u_h^\gamma], \frac{1}{2}\left(\partial_n v^+ + \partial_n v^-\right) - \gamma h_e^{-1}[v] \right\rangle_e \right) \\
&\quad - \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I\\ K=K^+}} \left\langle \partial_n u^+, v^+ \right\rangle_e + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I\\ K=K^-}} \left\langle \partial_n u^-, v^- \right\rangle_e \\
&= (f + \Delta u_h^\gamma, v)_K + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}_D^B}} \left\langle u_h^\gamma - g_D, \partial_n v - \gamma h_e^{-1} v \right\rangle_e + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}_N^B}} \left\langle g_N - \partial_n u_h^\gamma, v \right\rangle_e \\
&\quad + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I\\ K=K^+}} \left(\left\langle [u_h^\gamma], \frac{1}{2}\partial_n v^+ - \gamma h_e^{-1} v^+ \right\rangle_e - \frac{1}{2}\left\langle [\partial_n u_h^\gamma], v^+ \right\rangle_e \right) \\
&\quad + \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I\\ K=K^-}} \left(\left\langle [u_h^\gamma], \frac{1}{2}\partial_n v^- + \gamma h_e^{-1} v^- \right\rangle_e - \frac{1}{2}\left\langle [\partial_n u_h^\gamma], v^- \right\rangle_e \right).
\end{aligned}
\tag{3.22}
$$

3.3 Estimator Performance

3.3.1 Effectivity Index Comparison: Arnold vs. Baker

Figures 3.1–3.9 show $\|e\|$, $\|\nabla e\|$, and $\|e\|_{1,h}$ for functions f3, f4, and f6 under full refinement through 7 levels. Figures 3.10–3.15 show effectivity indices for the same set of runs.

For the residual estimator, the effectivity index η_1 is calculated as

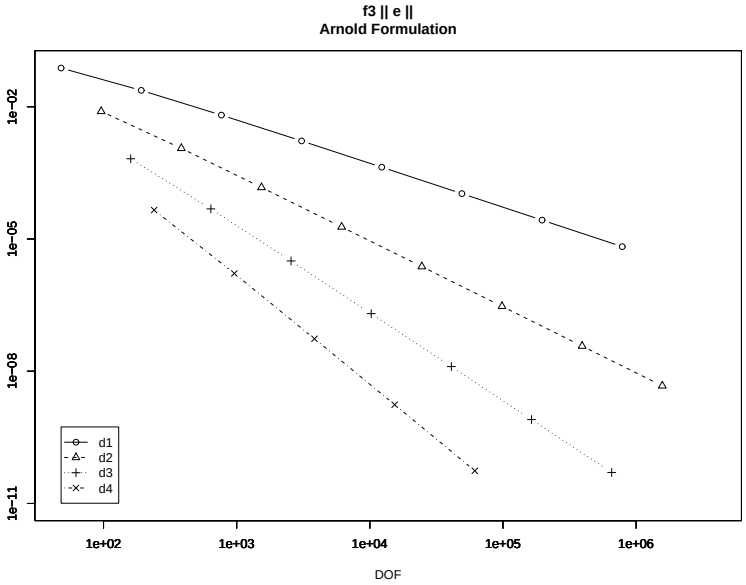
$$
\eta_1^2 = \frac{\xi^2}{\|e\|_{1,h}^2} \tag{3.23}
$$

where

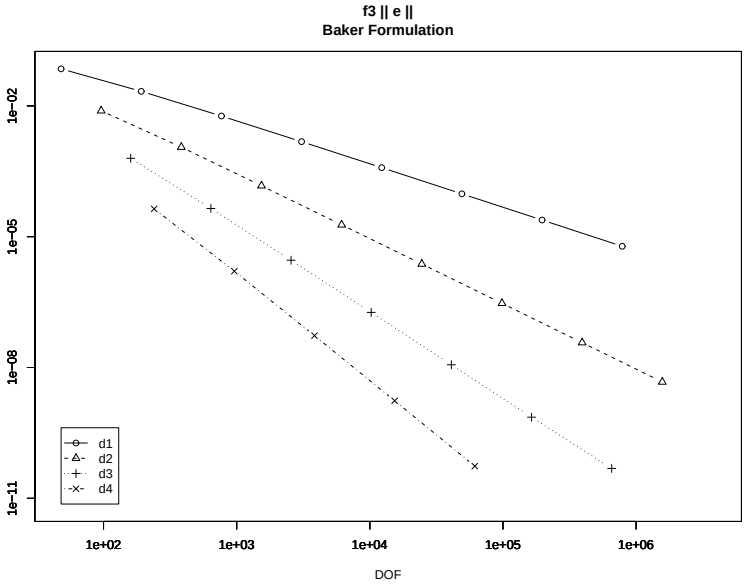
$$
\begin{aligned}
\xi^2 = \sum_{K\in\mathcal{T}_h} h_K^2 \|f + \Delta u_h^\gamma\|_K^2 + \sum_{e\in\mathcal{E}^I} h_e |[\partial_n u_h^\gamma]|_e^2 + \sum_{e\in\mathcal{E}_N^B} h_e |g_N - \partial_n u_h^\gamma|_e^2 \\
+ \sum_{e\in\mathcal{E}^I} \gamma^2 h_e^{-1} |[u_h^\gamma]|_e^2 + \gamma^2 \sum_{e\in\mathcal{E}_D^B} h_e^{-1} |g_D - u_h^\gamma|_e^2.
\end{aligned}
$$

For the local problem estimator, the effectivity index η_2 is calculated as

$$
\eta_2 = \frac{\|\eta\|_{1,h}}{\|e\|_{1,h}} \quad \text{where} \quad \eta = \left(\sum_{K\in\mathcal{T}_h} \eta_K^2 \right)^{1/2}. \tag{3.24}
$$

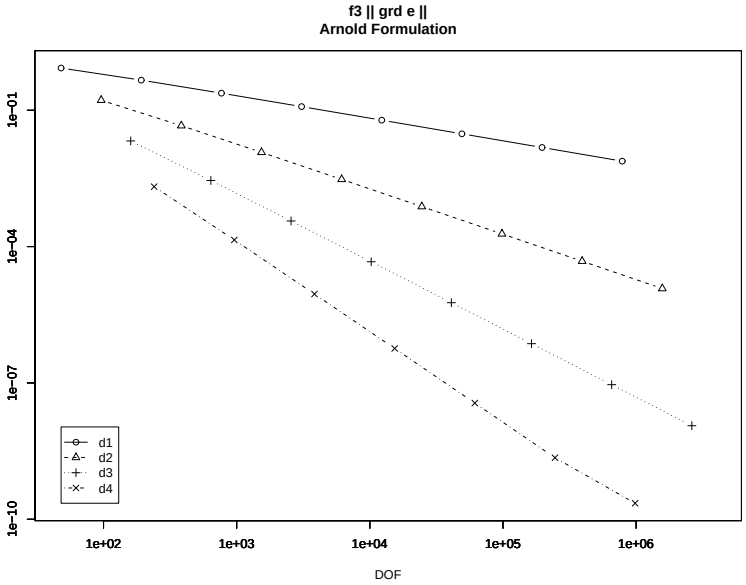


(a) f3 $||e||$ - Arnold

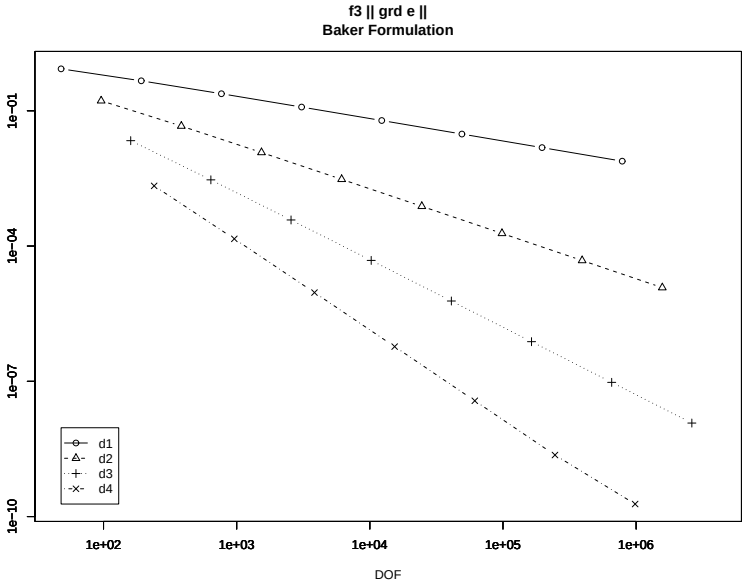


(b) f3 $||e||$ - Baker

Figure 3.1: f3 $||e||$

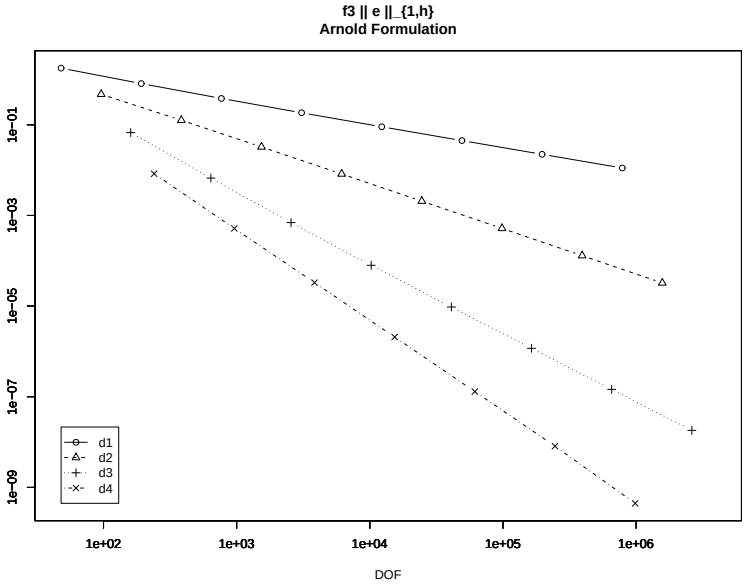


(a) f3 $||\nabla e||$ - Arnold

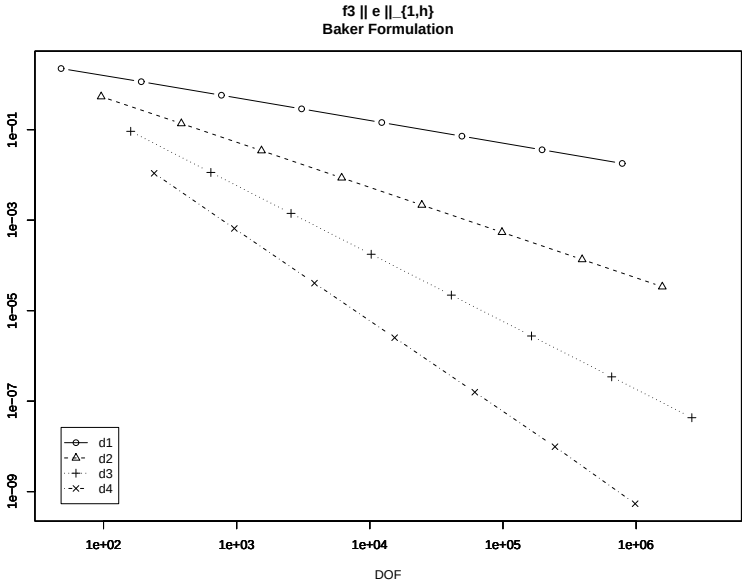


(b) f3 $||\nabla e||$ - Baker

Figure 3.2: f3 $||\nabla e||$

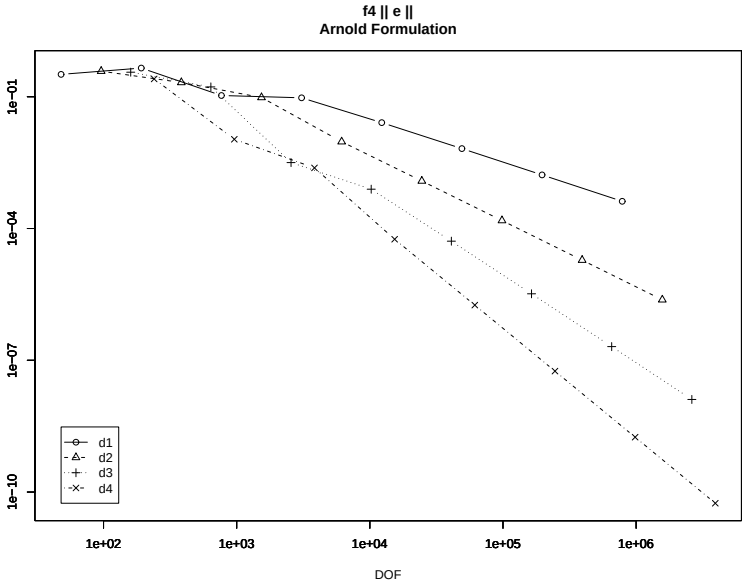


(a) f3 $||e||_{1,h}$ - Arnold

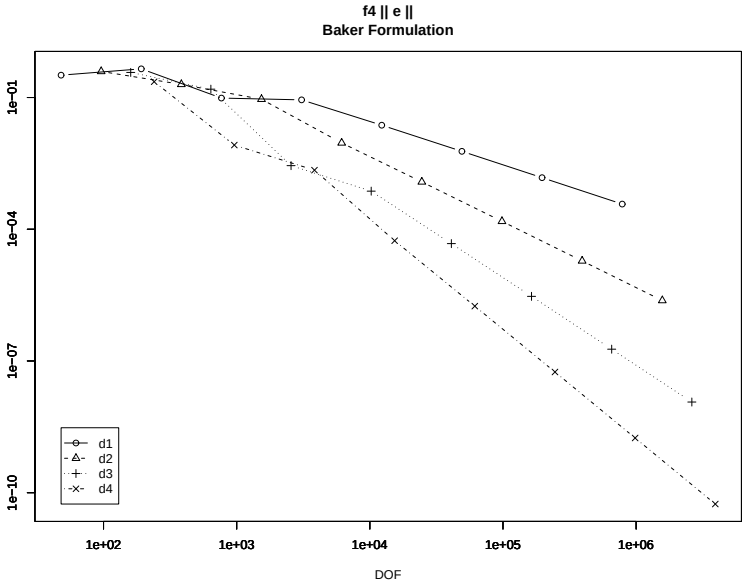


(b) f3 $||e||_{1,h}$ - Baker

Figure 3.3: f3 $||e||_{1,h}$

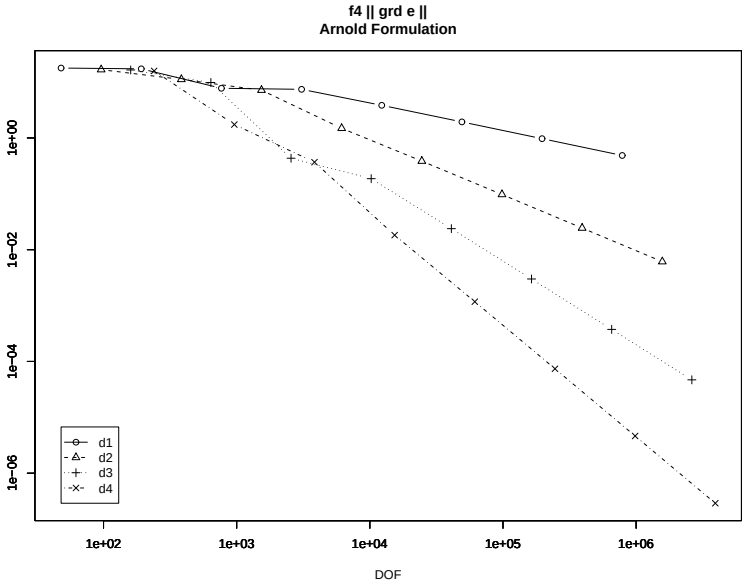


(a) f4 $||e||$ - Arnold

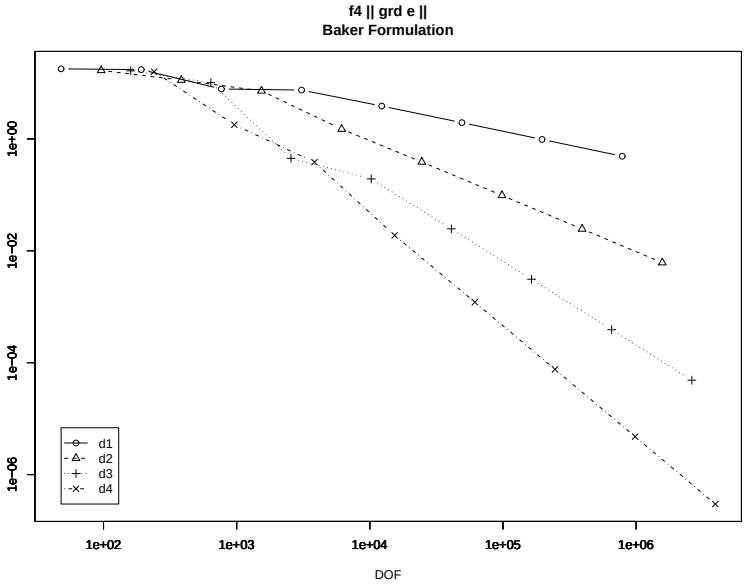


(b) f4 $||e||$ - Baker

Figure 3.4: f4 $||e||$

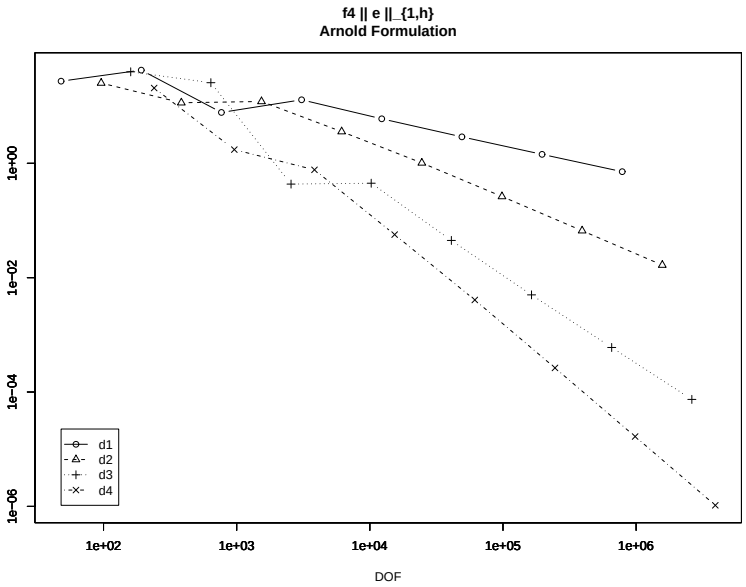


(a) f4 $||\nabla e||$ - Arnold

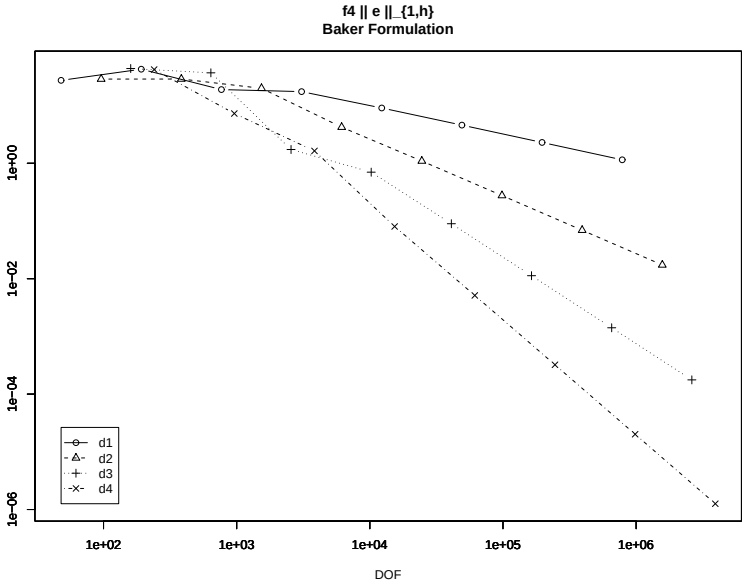


(b) f4 $||\nabla e||$ - Baker

Figure 3.5: f4 $||\nabla e||$

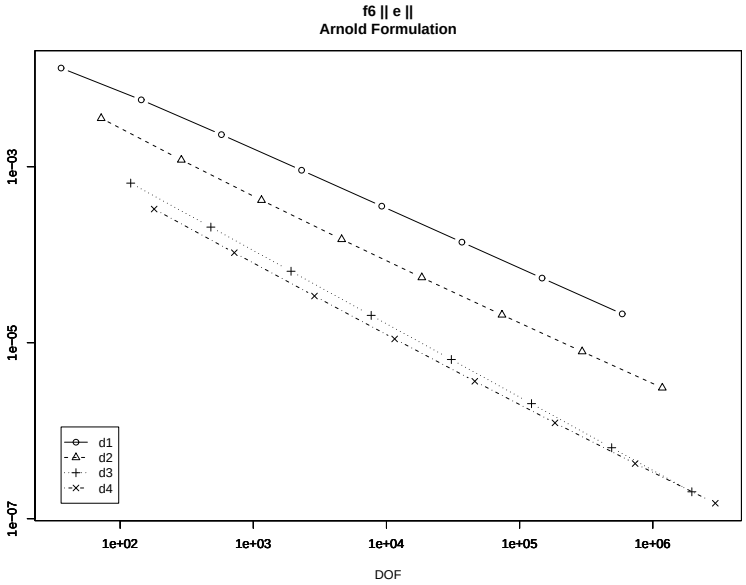


(a) f4 $||e||_{1,h}$ - Arnold

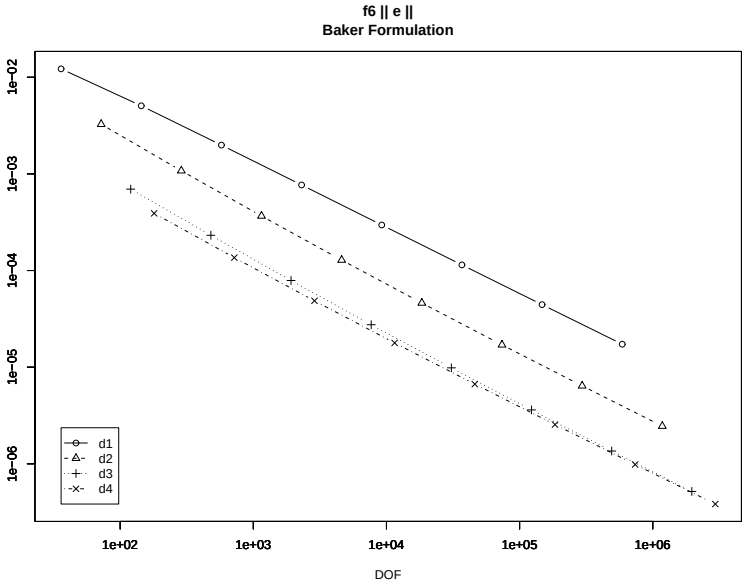


(b) f4 $||e||_{1,h}$ - Baker

Figure 3.6: f4 $||e||_{1,h}$

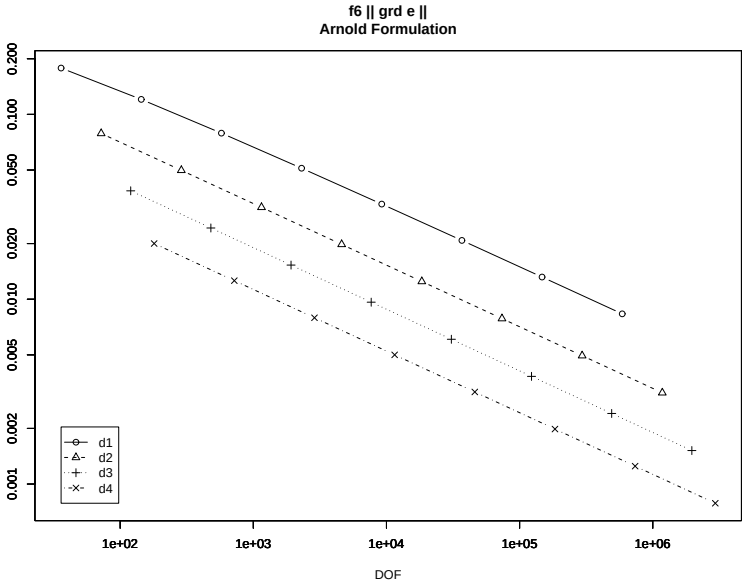


(a) f6 $||e||$ - Arnold

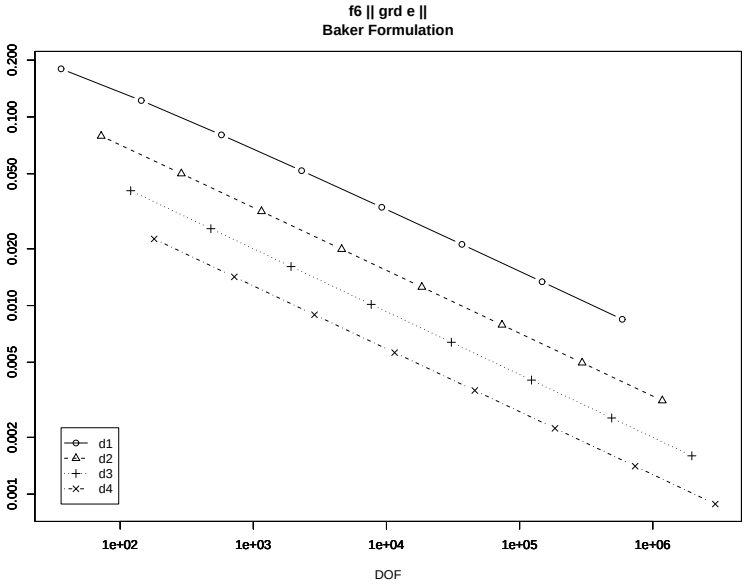


(b) f6 $||e||$ - Baker

Figure 3.7: f6 $||e||$

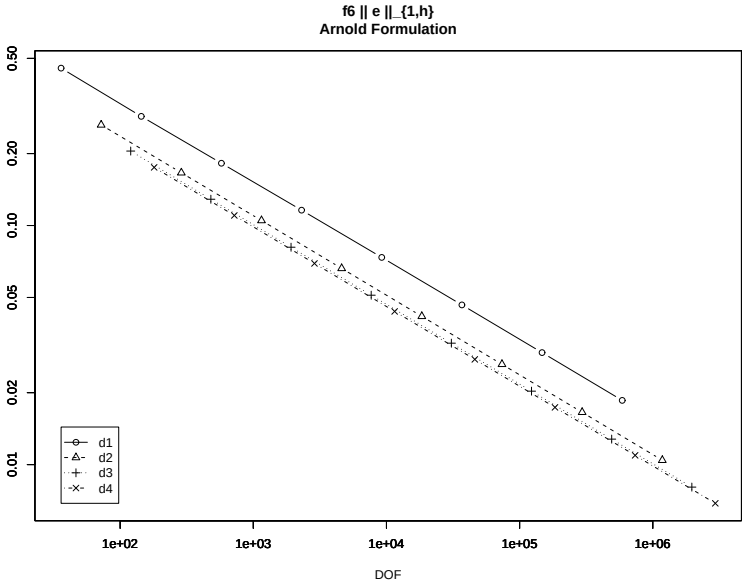


(a) f6 $||\nabla e||$ - Arnold

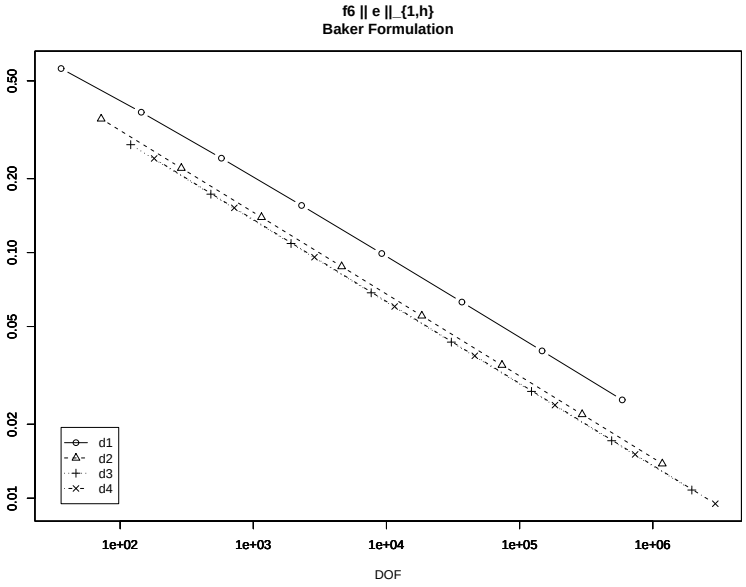


(b) f6 $||\nabla e||$ - Baker

Figure 3.8: f6 $||\nabla e||$

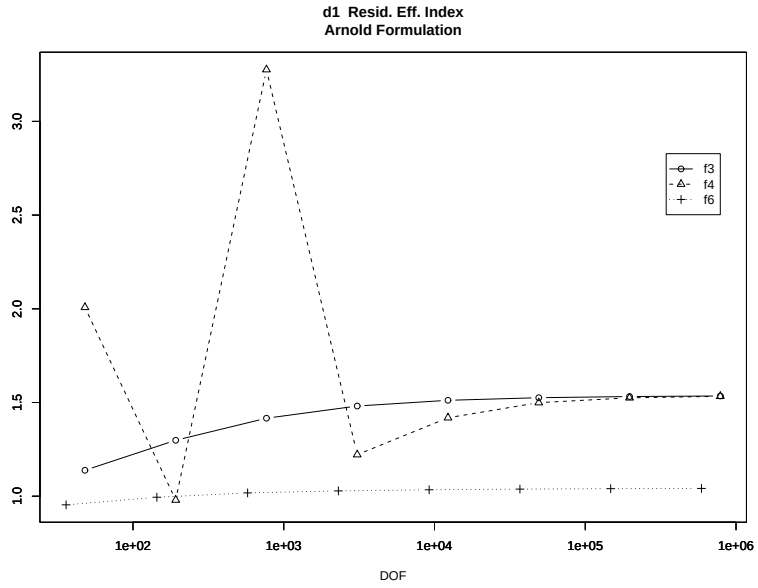


(a) f6 $||e||_{1,h}$ - Arnold

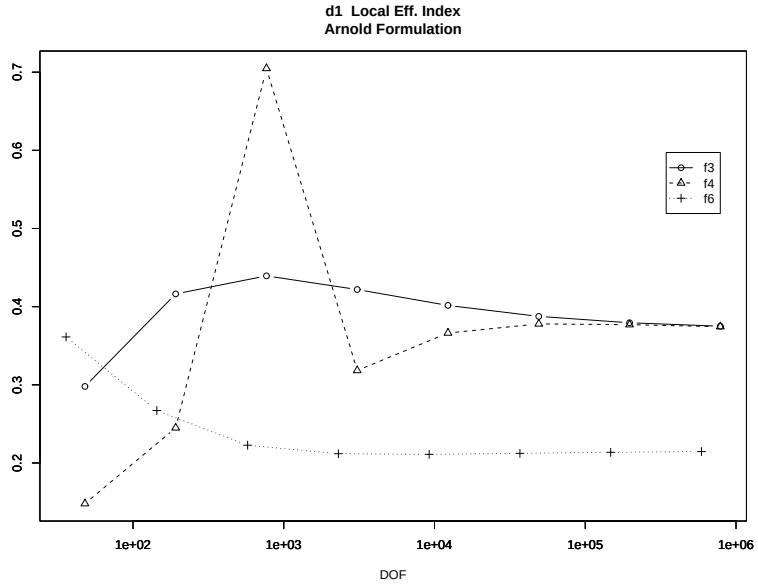


(b) f6 $||e||_{1,h}$ - Baker

Figure 3.9: f6 $||e||_{1,h}$

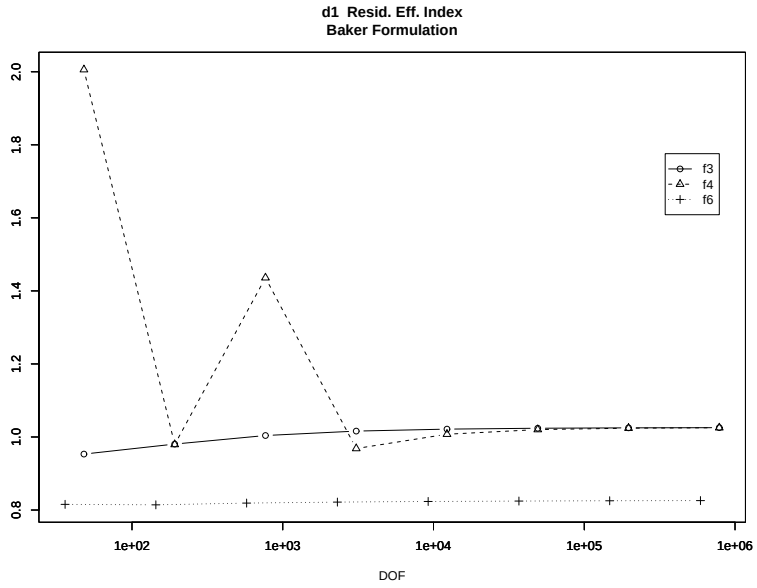


(a) Effectivity Index - $r = 2$, Arnold, Residual

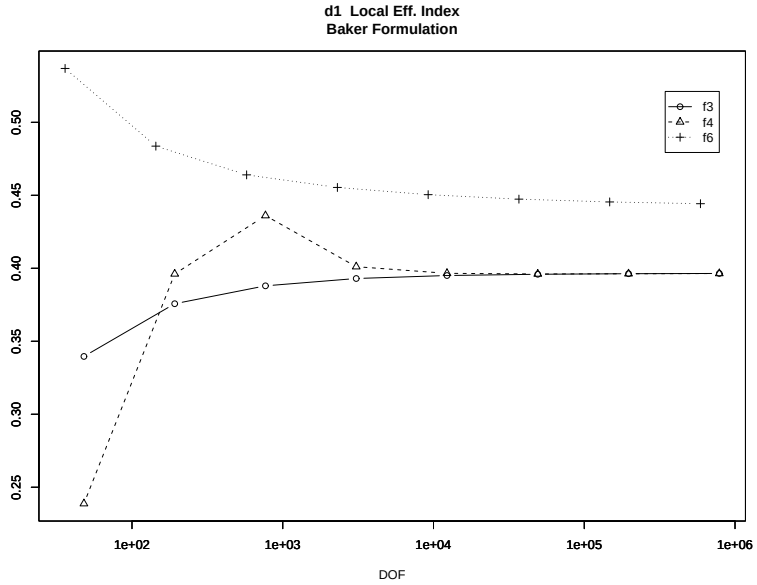


(b) Effectivity Index - $r = 2$, Arnold, Local

Figure 3.10: $r = 2$ Effectivity Indices, Arnold

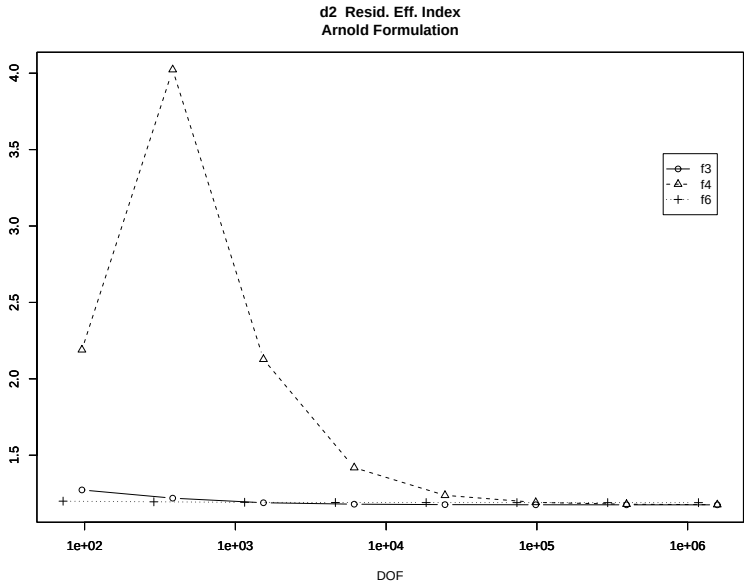


(a) Effectivity Index - $r = 2$, Baker, Residual

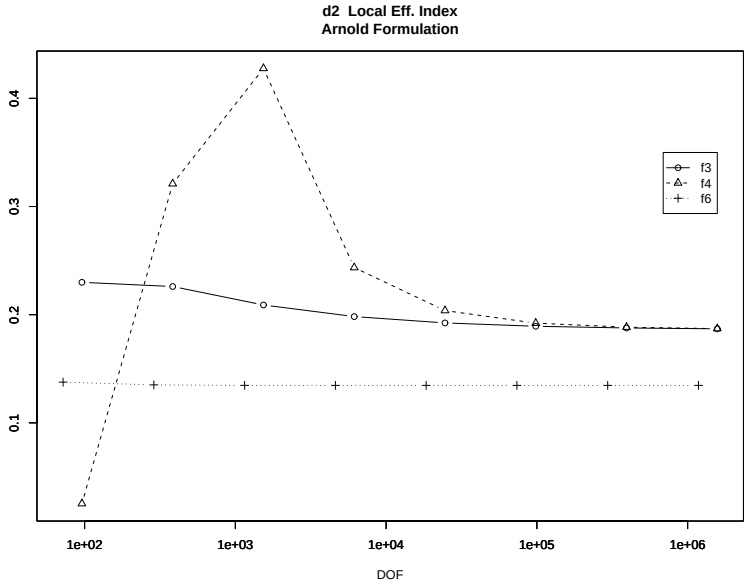


(b) Effectivity Index - $r = 2$, Baker, Local

Figure 3.11: $r = 2$ Effectivity Indices, Baker

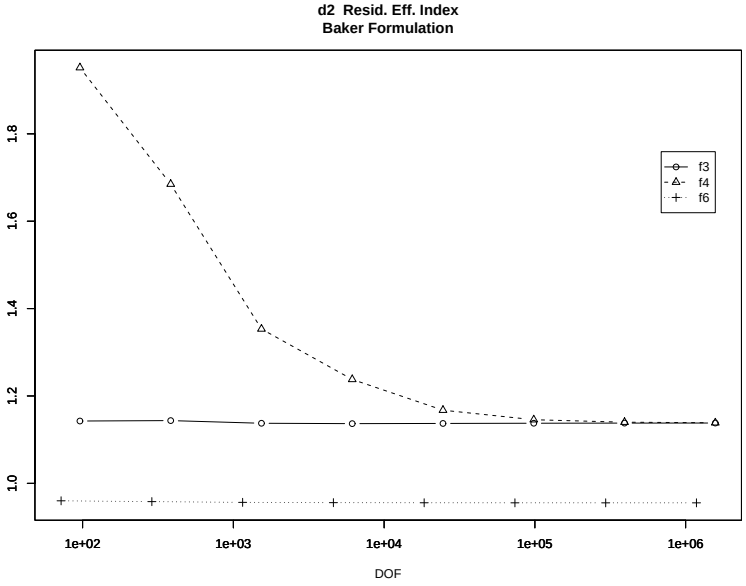


(a) Effectivity Index - $r = 3$, Arnold, Residual

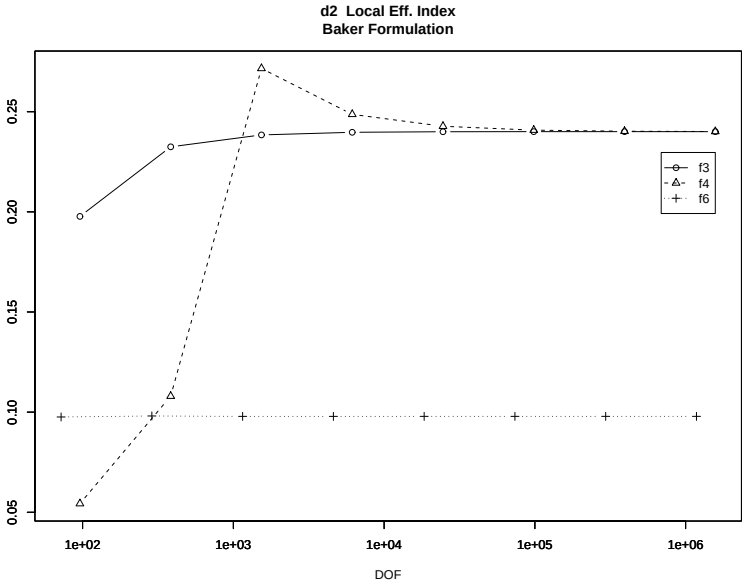


(b) Effectivity Index - $r = 3$, Arnold, Local

Figure 3.12: $r = 3$ Effectivity Indices, Arnold

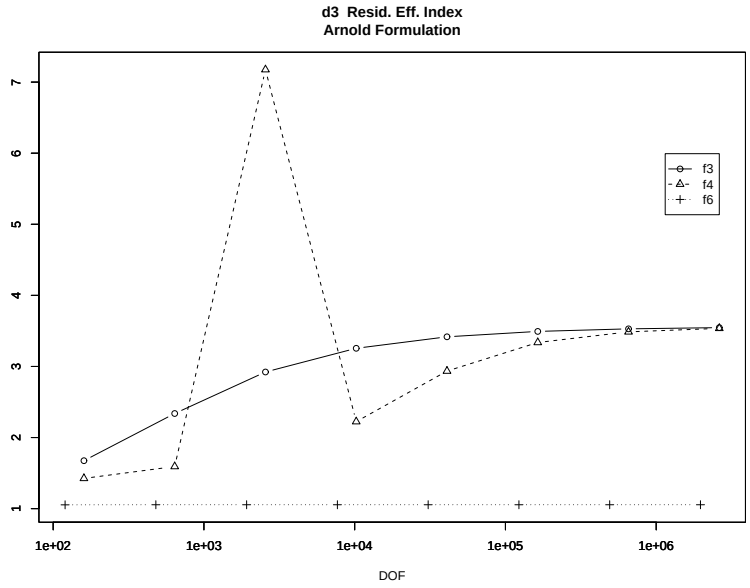


(a) Effectivity Index - $r = 3$, Baker, Residual

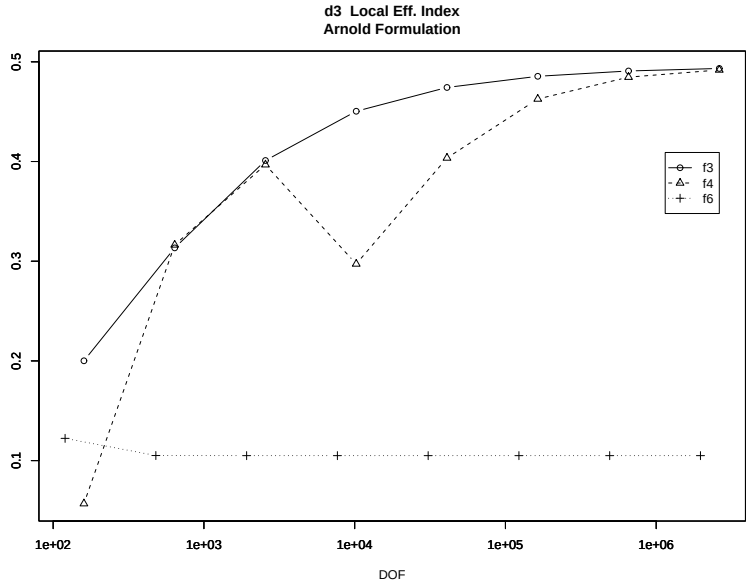


(b) Effectivity Index - $r = 3$, Baker, Local

Figure 3.13: $r = 3$ Effectivity Indices, Baker

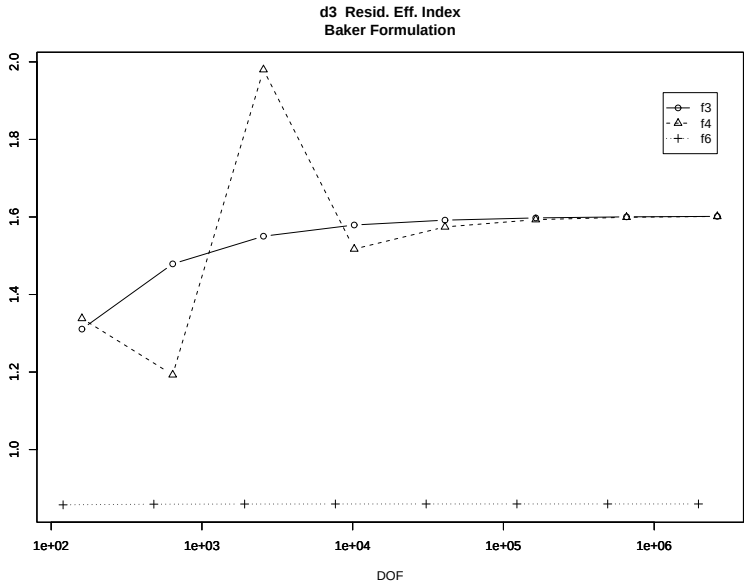


(a) Effectivity Index - $r = 4$, Arnold, Residual

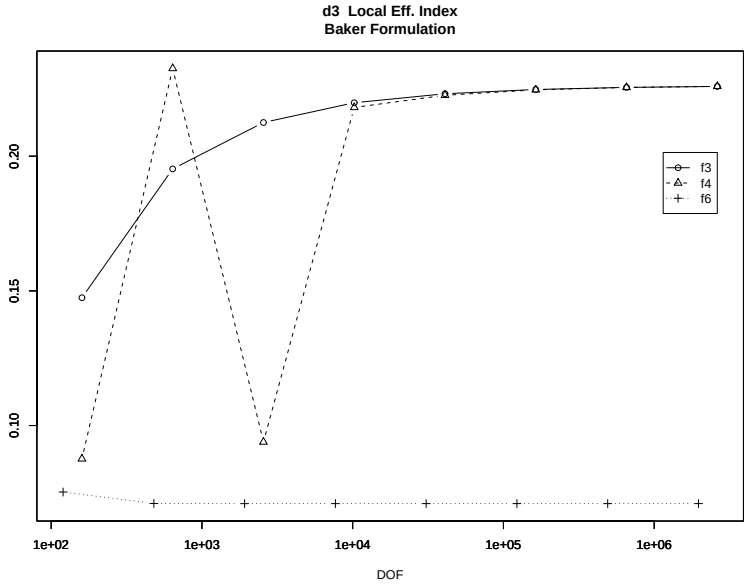


(b) Effectivity Index - $r = 4$, Arnold, Local

Figure 3.14: $r = 4$ Effectivity Indices, Arnold



(a) Effectivity Index - $r = 4$, Baker, Residual



(b) Effectivity Index - $r = 4$, Baker, Local

Figure 3.15: $r = 4$ Effectivity Indices, Baker

Note that in both cases, the following energy norm is used, which differs from Eq 3.11 in that γ influence is included:

$$\|v\|_{1,h} = \left(\sum_{K \in \mathcal{T}_h} \|\nabla v\|_{0,K}^2 + \sum_{e \in \mathcal{E}^I \cup \mathcal{E}_D^B} \left(\gamma h_e^{-1} |[v]|_{0,e}^2 + h_e \left| \{\partial_n v\} \right|_{0,e}^2 \right) \right)^{1/2} . \tag{3.25}$$

Some observations on the error norm graphs. First note that for the oscillatory test problem (f4), there is some initial bad behavior. This is due to the fact that the initial mesh was too coarse to obtain accurate approximations. As the mesh becomes more refined, this bad behavior is eliminated and one ends up with the predicted a priori error norm reductions. Second, the poor behavior for high degree polynomials on highly refined meshes is not yet clearly understood. For $h \to h/2$ uniformly in a mesh with elements of degree p, one expects that

- $\|u - u_{h/2}\|_{L^2(\Omega)} \approx \left(\frac{1}{2}\right)^{p+1} \|u - u_h\|_{L^2(\Omega)}$
- $\|u - u_{h/2}\|_{H^1(\Omega)} \approx \left(\frac{1}{2}\right)^{p} \|u - u_h\|_{H^1(\Omega)}$

As one can see in the following graphs, this is indeed the case for the smooth functions (f3, f4), but not necessarily the case for the point singularity problem (f6).

Some observations on the effectivity indices. The effectivity indices show good behavior in that they are generally close to 1, indicating that the estimator predicts the error fairly well. There also seems to be some dependence on r.

3.3.2 Adaptive Performance Comparison: Arnold vs. Baker

The following figures provide insight into how the adaptive methods perform for various problems. The reduction in energy norm for f3, f4, and f6 are given followed by the adaptive meshes produced for each $r = 2, 3, 4$. As one can see, the number of adaptive iterations required to reach the adaptive tolerance is very good.

Some observations on the meshes. For the residual estimator, Arnold and Baker formulations give very similar meshes. For the local estimator, Arnold and Baker formulations give different meshes, but move towards better agreement with the residual estimator and with each other as the polynomial degree $r - 1$ increases.

One should note though that quality of the mesh picture does not necessarily imply anything about how well the error estimators performed. Note that there appears to be differences between the local estimator and the residual estimator for $r = 2, 3$, yet this difference seems to not be as obvious for $r = 4$. To obtain a better picture of the local estimator, it would be worthwhile to do both h and p refinement of the V' subspace, which might provide even better results.

Table 3.1 illustrates the critical parameters used in generating these meshes. Note that for each set of problem meshes by r for problems f3,f4, and f6 the error in the H^1 seminorm $\|\nabla e\|$ are approximately the same. We could not do this for problem f7 since we do not have the exact solution, and so these meshes are provided just to illustrate refinement behavior near the corners.

Table 3.1: `E112` Adaptive Runs

run	γ	ϵ_{adapt}	dof	vertices	tri	aIter	Lvl	$\|\nabla e\|$	Figure
f3d1AR_A	5	0.088	29046	5085	9682	5	5	0.040404	3.17(a)
f3d1AR_B	10	0.0942	32718	5746	10906	5	5	0.040424	3.17(b)
f3d1AL_A	5	0.0332	29262	5468	9754	5	5	0.0404795	3.17(c)
f3d1AL_B	10	0.0393	67998	16557	22666	8	7	0.0404051	3.17(d)
f3d2AR_A	4	0.001655	50496	4543	8416	5	5	0.00051552	3.19(a)
f3d2AR_B	7	0.001668	53394	4946	8899	5	5	0.000515721	3.19(b)
f3d2AL_A	4	0.0002454	55878	5253	9313	6	5	0.000515185	3.19(c)
f3d2AL_B	7	0.000221	64698	6192	10783	6	5	0.000515457	3.19(d)
f3d3AR_A	4	1.93e-05	98140	5299	9814	5	5	3.51251e-06	3.21(a)
f3d3AR_B	6	1.985e-05	102220	5715	10222	5	5	3.51317e-06	3.21(b)
f3d3AL_A	4	2.794e-06	101260	5607	10126	5	5	3.5146e-06	3.21(c)
f3d3AL_B	6	2.21e-06	108850	6506	10885	5	5	3.5185e-06	3.21(d)
f4d1AR_A	6	4.42	42258	7875	14086	6	5	2.07931	3.23(a)
f4d1AR_B	10	4.848	46353	8225	15451	6	5	2.08712	3.23(b)
f4d1AL_A	6	1.493	40044	7435	13348	6	6	2.07948	3.23(c)
f4d1AL_B	10	2.2	102171	23453	34057	10	7	2.07492	3.23(d)
f4d2AR_A	4	0.518	75804	7003	12634	5	5	0.156587	3.25(a)
f4d2AR_B	7	0.534	77352	7101	12892	5	5	0.156213	3.25(b)
f4d2AL_A	4	0.09111	78234	7380	13039	6	6	0.157095	3.25(c)
f4d2AL_B	7	0.091	86082	8026	14347	6	6	0.156146	3.25(d)
f4d3AR_A	4	0.0764	99250	5775	9925	5	5	0.0146416	3.27(a)
f4d3AR_B	6	0.07896	103300	6185	10330	5	5	0.0146036	3.27(b)
f4d3AL_A	4	0.01095	100720	5989	10072	5	5	0.0147155	3.27(c)
f4d3AL_B	6	0.009	107230	6603	10723	5	5	0.0147023	3.27(d)
f6d1AR_A	7	0.0184	19062	3394	6354	9	9	0.00938087	3.29(a)
f6d1AR_B	10	0.0206	25182	4597	8394	9	9	0.00933783	3.29(b)
f6d1AL_A	7	0.00838	24471	4606	8157	8	8	0.00930545	3.29(c)
f6d1AL_B	10	0.00902	43416	9879	14472	12	11	0.00932394	3.29(d)
f6d2AR_A	6	0.00117	34596	3370	5766	14	14	0.000349636	3.31(a)
f6d2AR_B	7	0.00122	42984	4029	7164	14	13	0.000347114	3.31(b)
f6d2AL_A	6	0.00019	35478	3306	5913	13	13	0.000347442	3.31(c)
f6d2AL_B	7	0.0002	37998	3779	6333	15	13	0.000348594	3.31(d)
f6d3AR_A	5	0.000154	30120	1786	3012	18	18	2.98224e-05	3.33(a)
f6d3AR_B	6	0.00016	32610	1917	3261	18	17	2.9442e-05	3.33(b)
f6d3AL_A	5	1.815e-05	47700	2722	4770	17	16	2.99311e-05	3.33(c)
f6d3AL_B	6	1.717e-05	33480	1916	3348	18	17	2.77807e-05	3.33(d)
f7d1AR_A	7	0.02	23952	4182	7984	7	7	-	3.34(a)
f7d1AR_B	10	0.02	39135	6966	13045	7	7	-	3.34(b)
f7d1AL_A	7	0.009	23196	4129	7732	7	7	-	3.34(c)
f7d1AL_B	10	0.009	82938	18032	27646	10	10	-	3.34(d)
f7d2AR_A	6	0.002	15360	1520	2560	9	8	-	3.35(a)
f7d2AR_B	7	0.002	12102	1170	2017	8	8	-	3.35(b)
f7d2AL_A	6	0.0005	11724	1139	1954	9	8	-	3.35(c)
f7d2AL_B	7	0.0005	10860	1112	1810	9	8	-	3.35(d)

Table 3.1, continued

run	γ	ϵ_{adapt}	dof	vertices	tri	aIter	Lvl	$\|\nabla e\|$	Figure
f7d3AR_A	5	0.0002	14470	883	1447	11	11	-	3.36(a)
f7d3AR_B	6	0.0002	16360	998	1636	11	11	-	3.36(b)
f7d3AL_A	5	5e-05	10120	615	1012	11	10	-	3.36(c)
f7d3AL_B	6	5e-05	10030	610	1003	11	10	-	3.36(d)

Note also that for the local problem estimator, η_K is determined on each element. The estimator then used as a refinement indicator is $\|\nabla \eta_K\|$. The apparent chopiness in the meshes for the Baker formulation of the local estimator appear to be related to the fact that our estimator is not including edge effects but rather is based soley on the gradient of the local problem solution η_K. Finally, note also that the refinement level for singularity problems f6 and f7 in general increases with degree, i.e., the mesh hierarchy gets deeper.

3.4 Penalty Term Study

One of the most cited difficulties with using DG SIPG methods is choosing a good value for the penalty parameter γ. It appears that using the Arnold formulation does a better job in predicting the error, improving with increasing r. Ideally, one would hope to be able to have a set of heuristic rules which would allow one to efficiently choose the value of γ for a particular problem. One should keep in mind that the theory only requires that γ be large enough to ensure coercivity of the bilinear form. For very large values of γ, the global stiffness matrix condition deteriorates and thus it is important to choose γ correctly. Finally, it should be noted that γ influence appears as $\gamma_c(r-1)^2$ in the bilinear form and appears as $\gamma_c^2(r-1)^4$ in the a posteriori residual based estimator, where γ_c is constant. This dependence of γ on r was shown in Karakashian and Pascal (2003).

3.4.1 Arnold and Baker Formulation: Error Norms

A comparison of Arnold and Baker formulations of $\|\nabla e\|$ for test problems f3,f4, and f6 while varying the penalty term γ is shown in Figures 3.37–3.42. One important point to note here is that for large γ, the Arnold and Baker formulations converge to the same $\|\nabla e\|$, which by theory should also be the same $\|\nabla e\|$ for the continuous Galerkin FEM formulation. Another point that needs to be made is that there exists a γ for which the minimum of $\|\nabla e\|$ is obtained and this γ is different for the Arnold and Baker formulations when working with the same problem and r.

3.4.2 Effectivity Indices

Effectivity indices for test problems f3, f4, f6 using both the Arnold and Baker formulations with the residual based estimator while varying the penalty term γ are compared in Figures 3.43– 3.54. As one can see, there is variation of the effectivity indices with γ, which implies that the efficiency of the estimator can vary. However, this variation is not so much that reasonable results cannot be obtained when using γ in a fairly wide range.

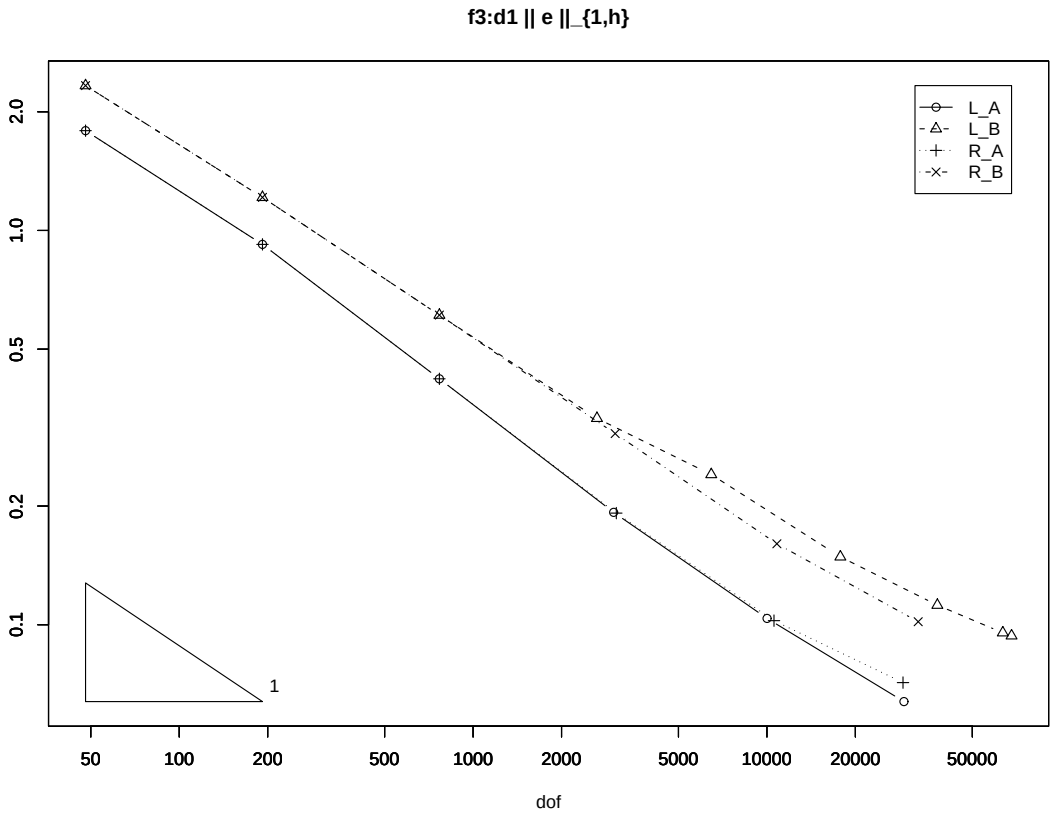
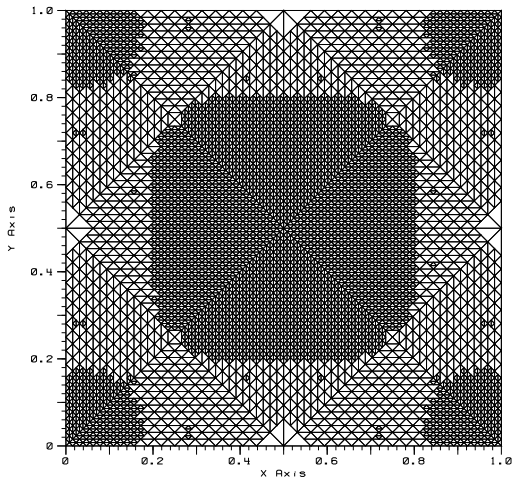
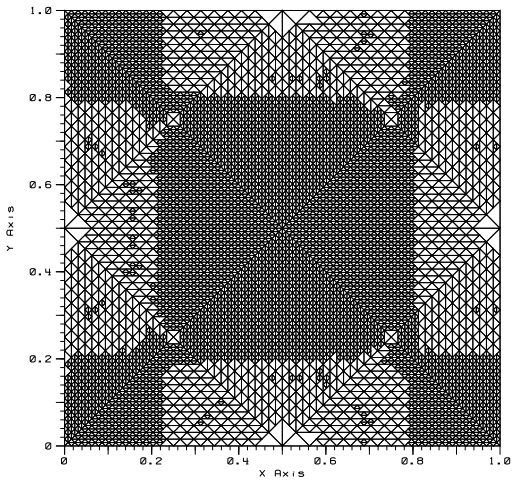


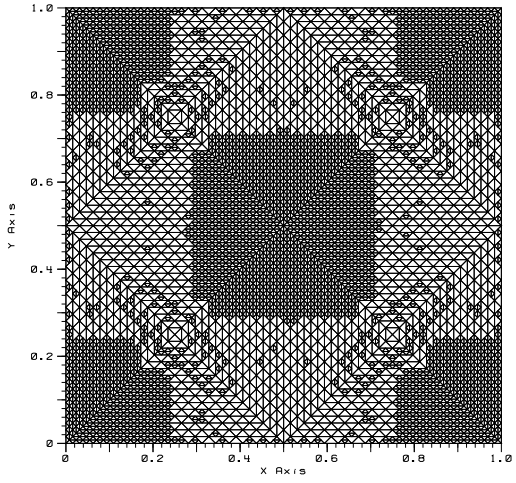
Figure 3.16: Adaptive $||e||_{1,h}$, f3, $r = 2$



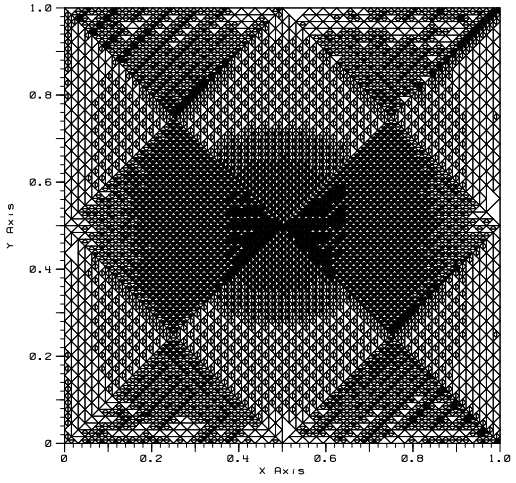
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.17: Adaptive Meshes: f3, r=2

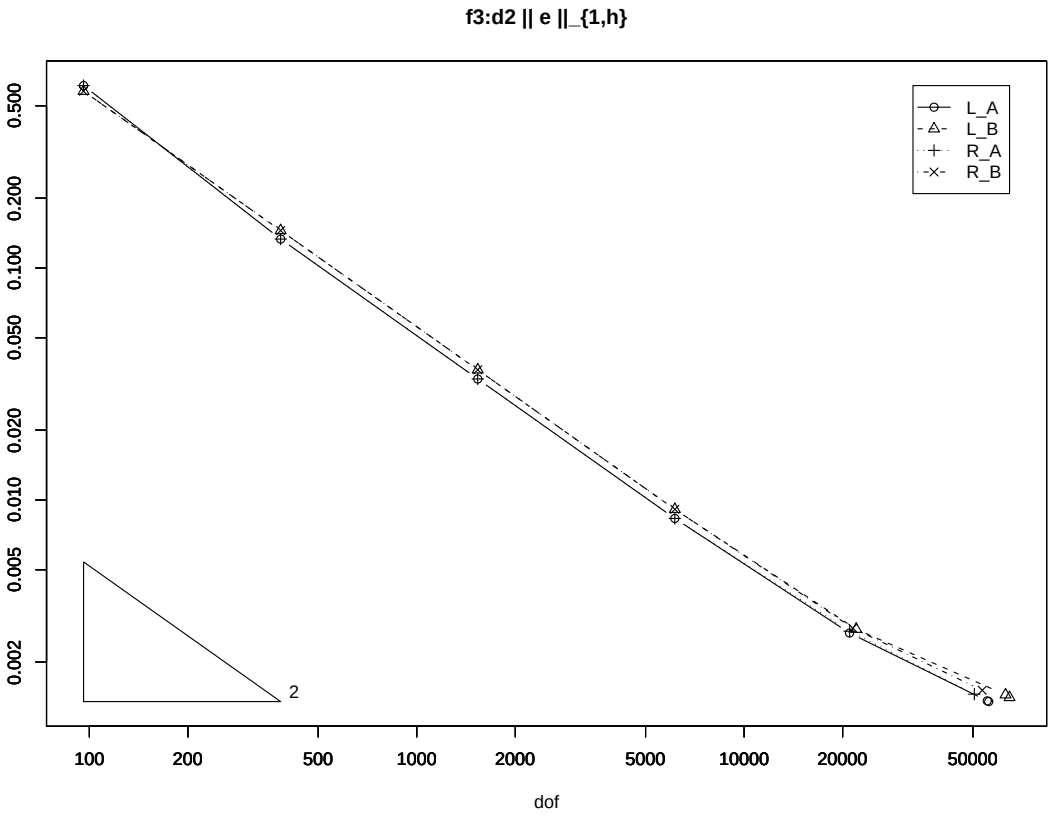
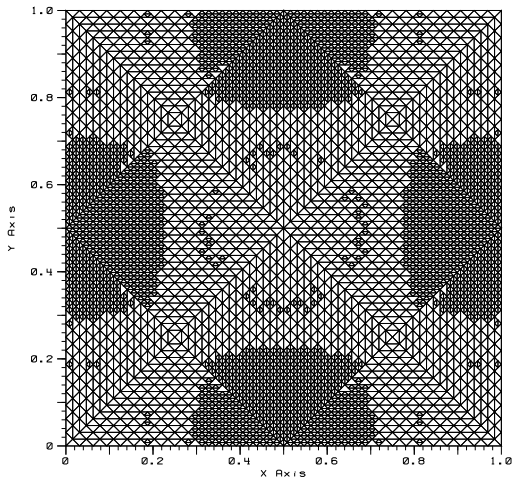
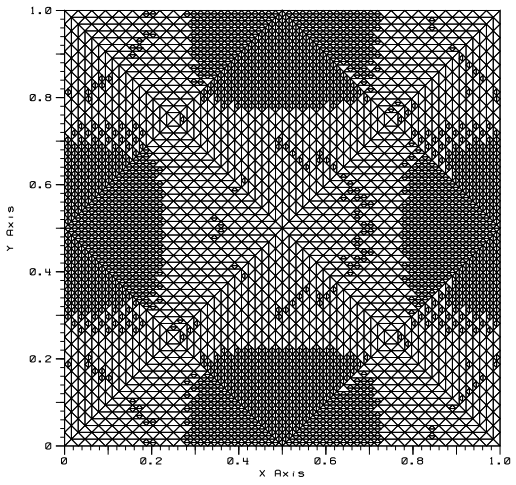


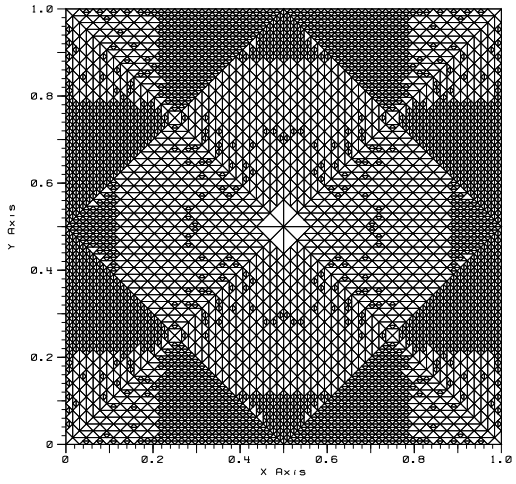
Figure 3.18: Adaptive $||e||_{1,h}$, f3, $r = 3$



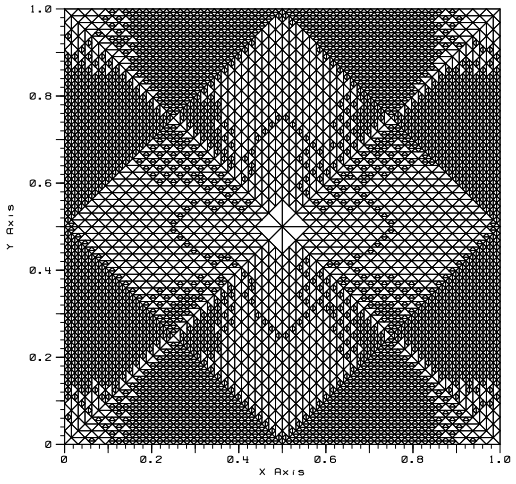
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.19: Adaptive Meshes: f3, r=3

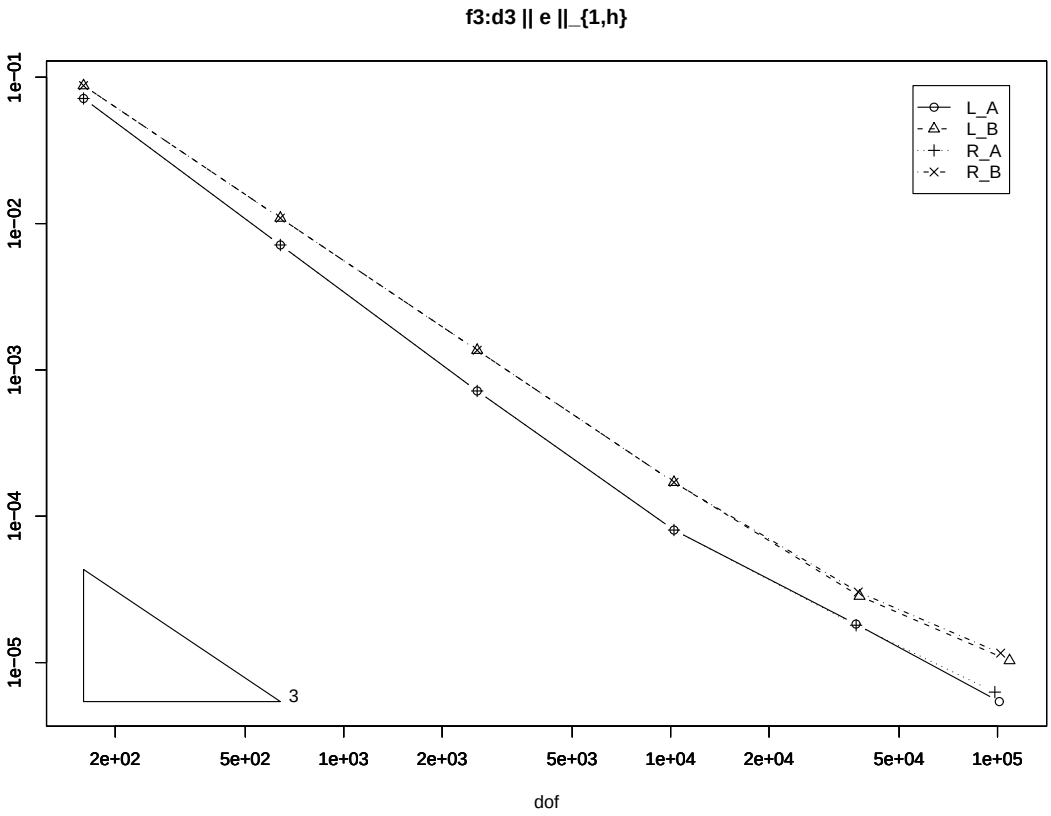
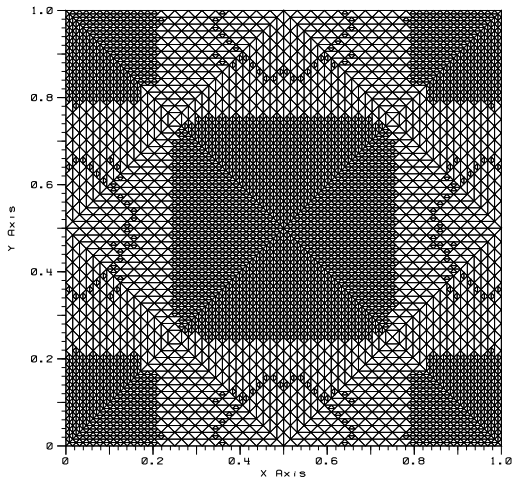
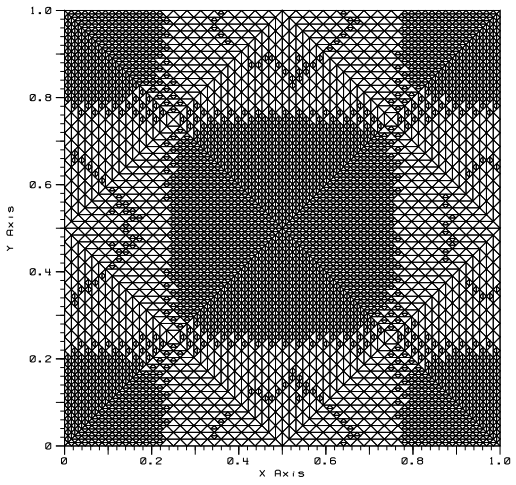


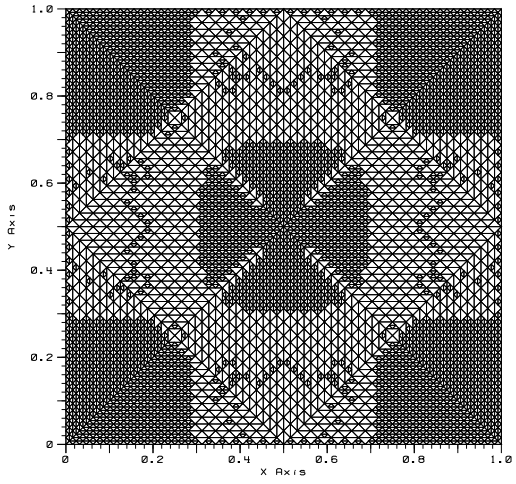
Figure 3.20: Adaptive $||e||_{1,h}$, f3, $r = 4$



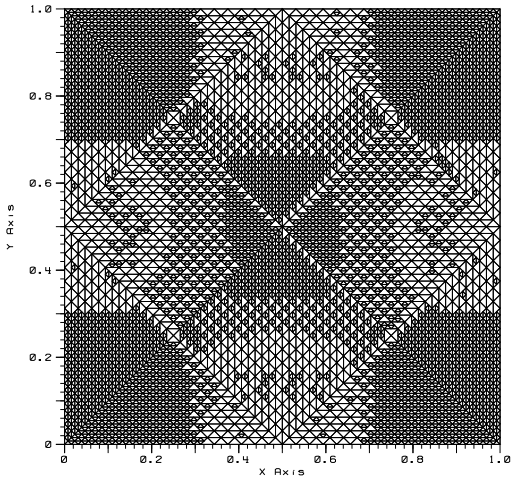
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.21: Adaptive Meshes: f3, r=4

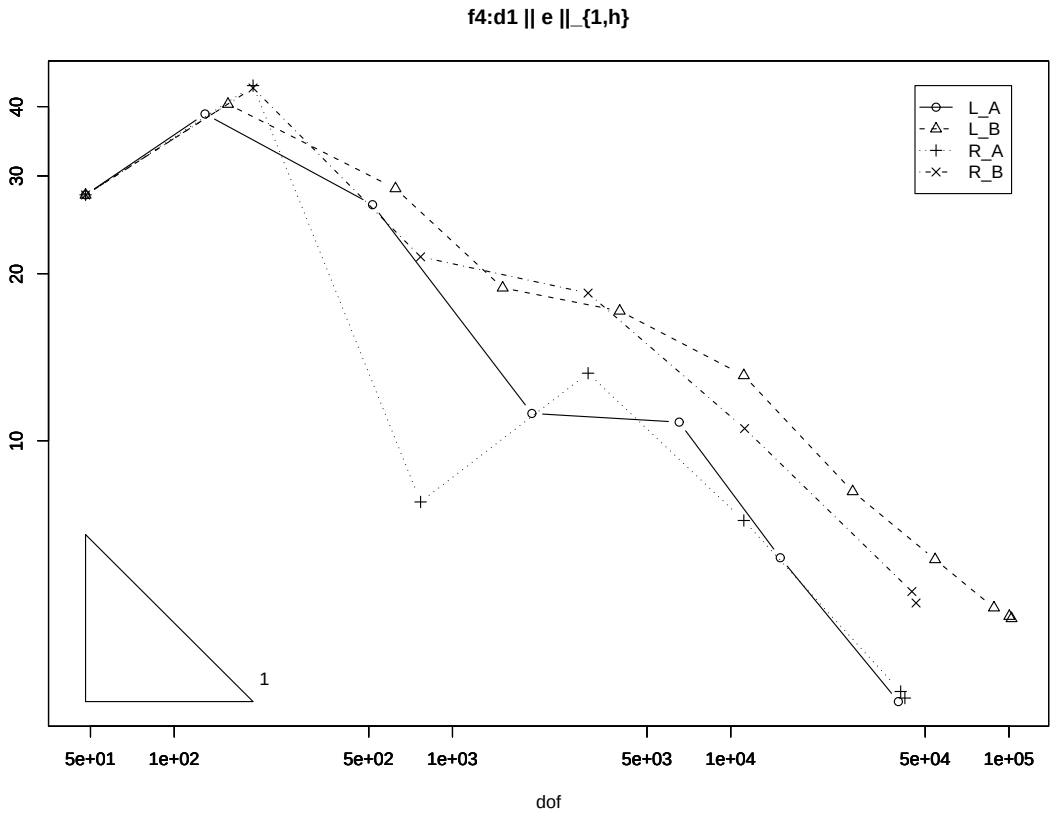
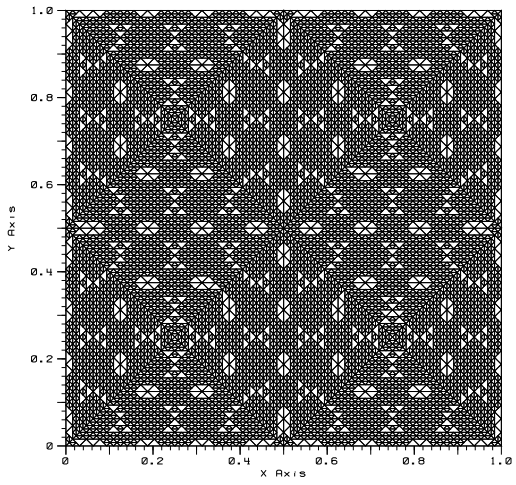
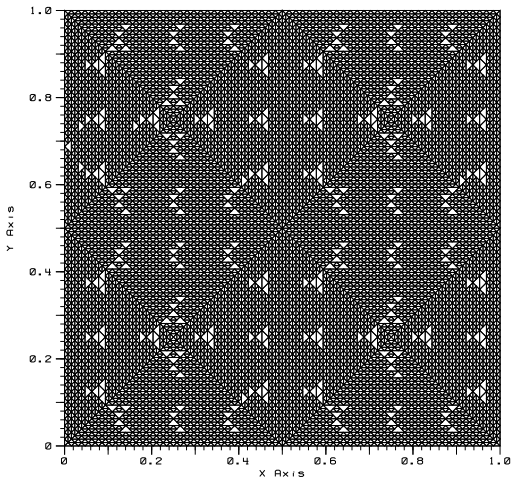


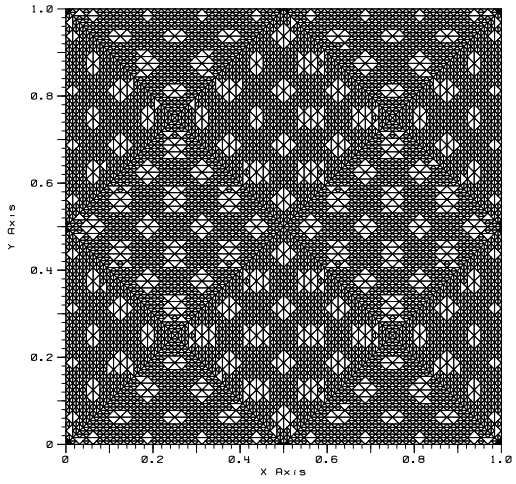
Figure 3.22: Adaptive $||e||_{1,h}$, f4, $r = 2$



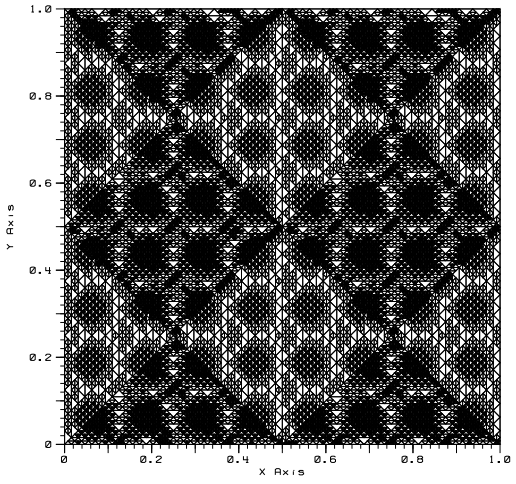
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.23: Adaptive Meshes: f4, r=2

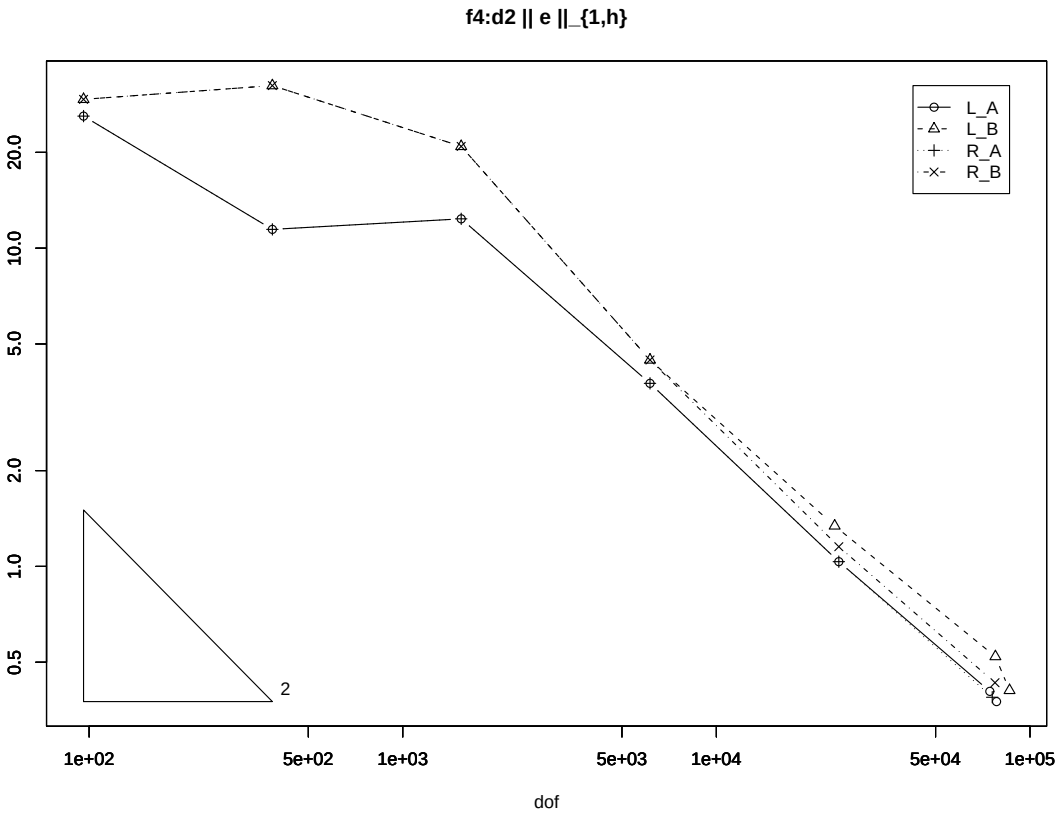
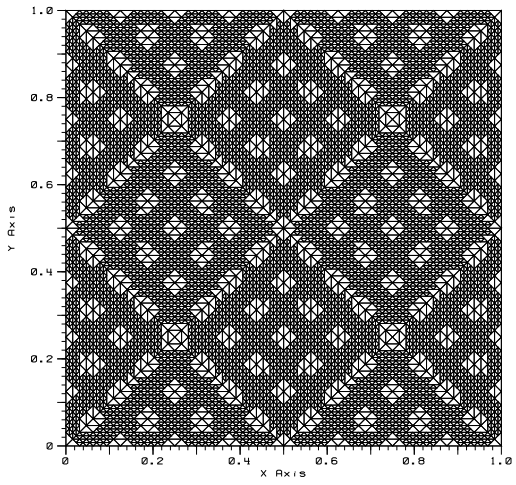
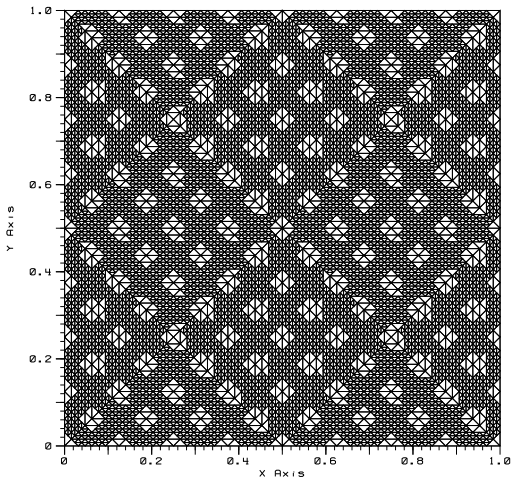


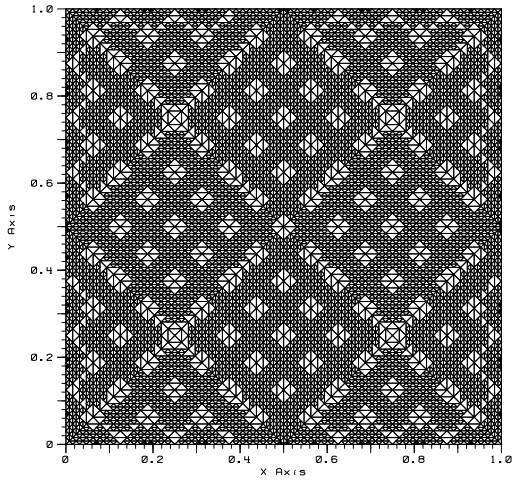
Figure 3.24: Adaptive $||e||_{1,h}$, f4, $r = 3$



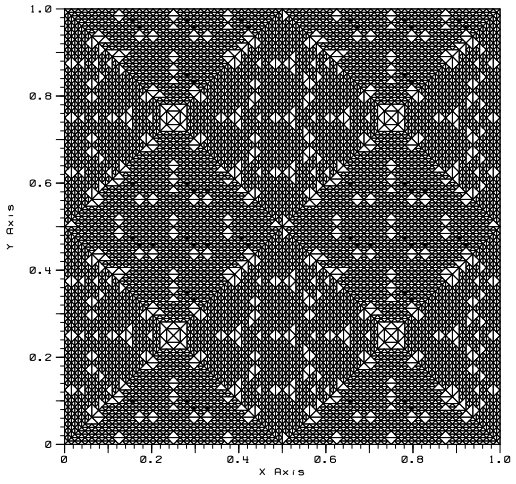
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.25: Adaptive Meshes: f4, r=3

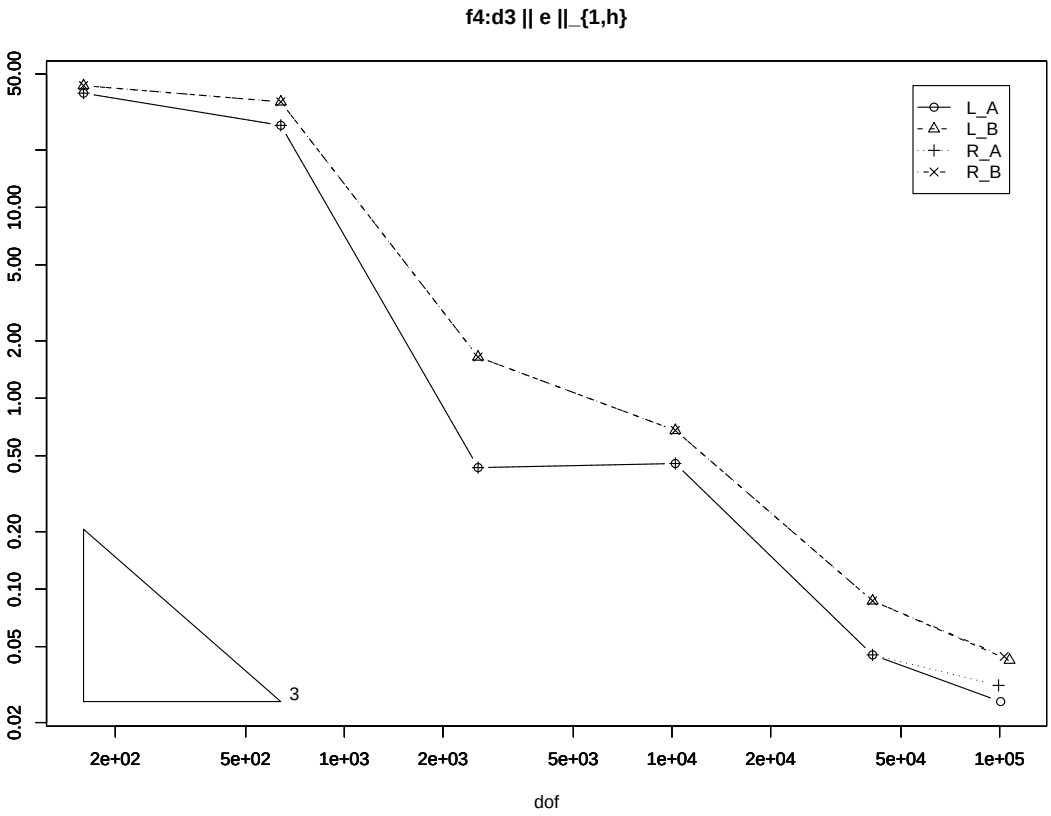
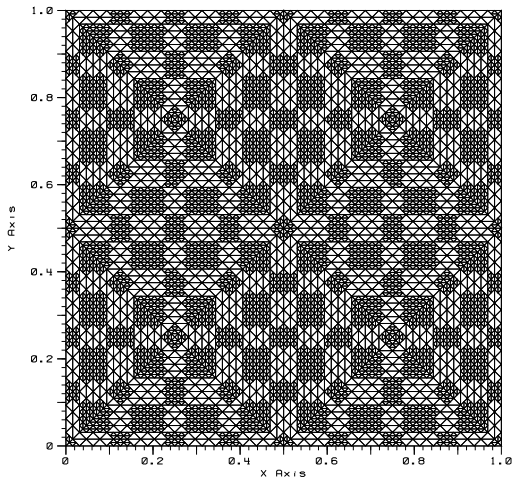
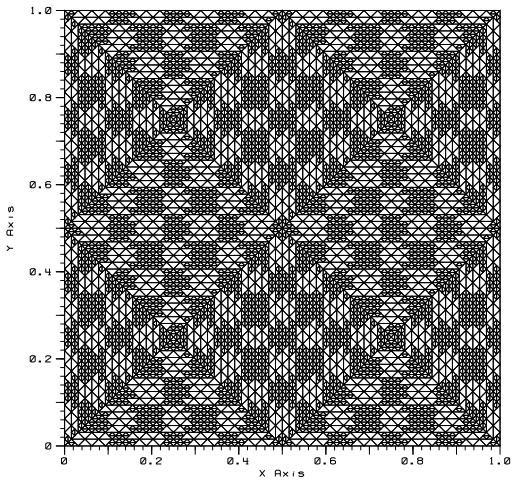


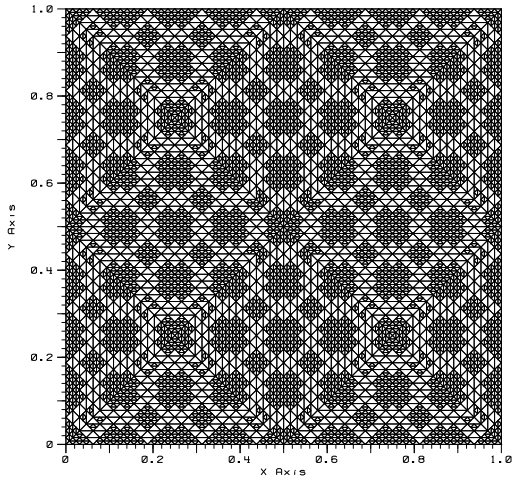
Figure 3.26: Adaptive $||e||_{1,h}$, f4, $r = 4$



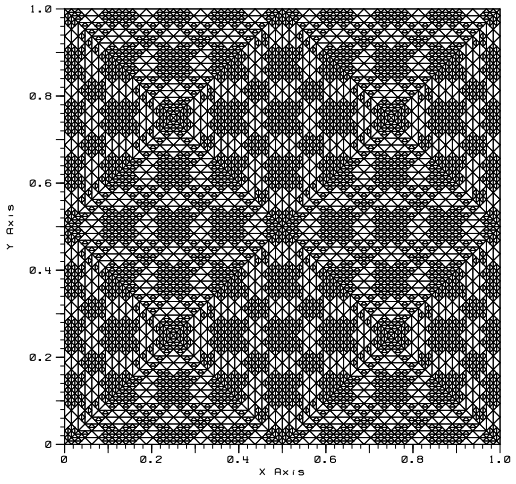
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.27: Adaptive Meshes: f4, r=4

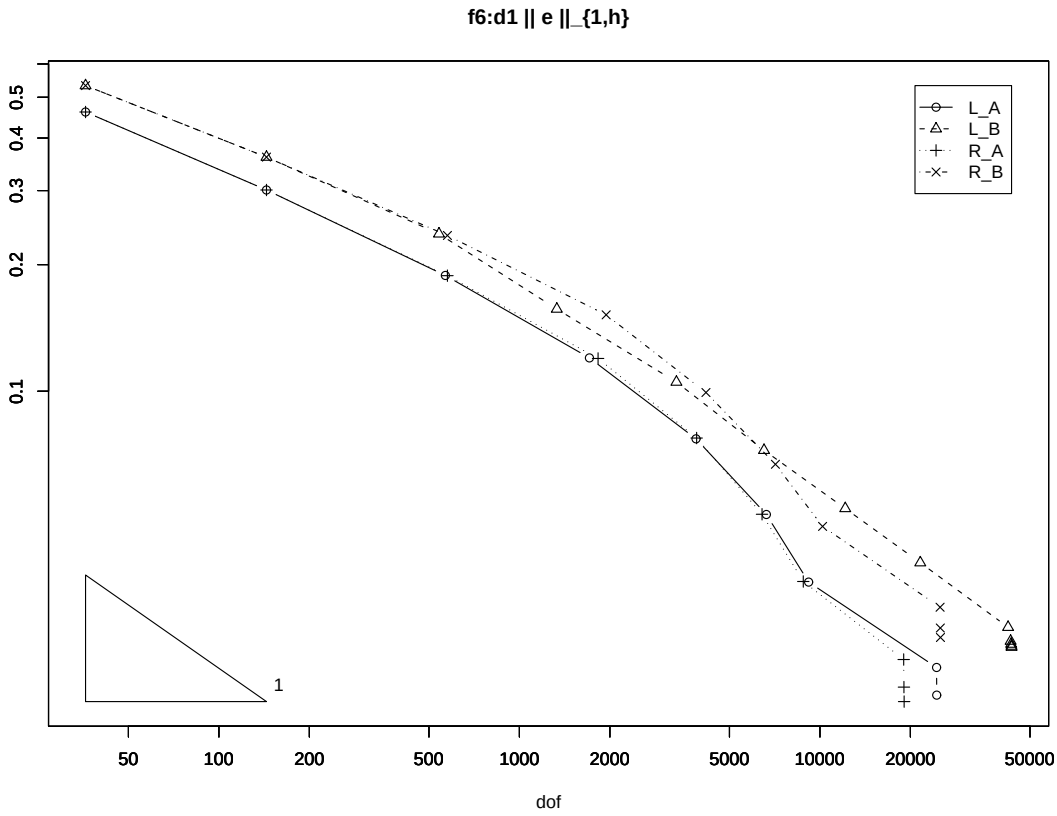
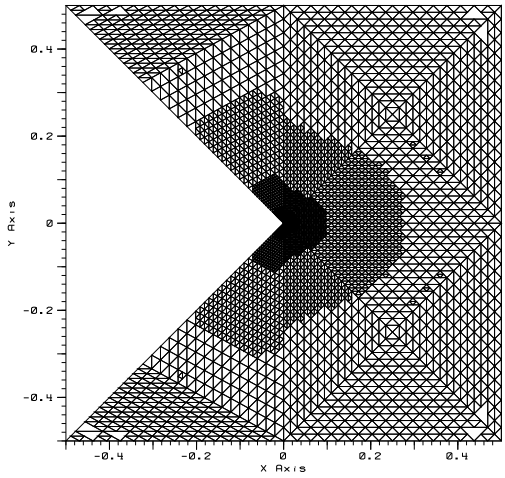
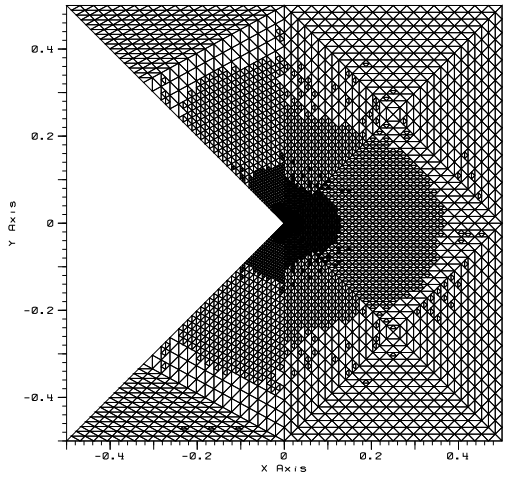


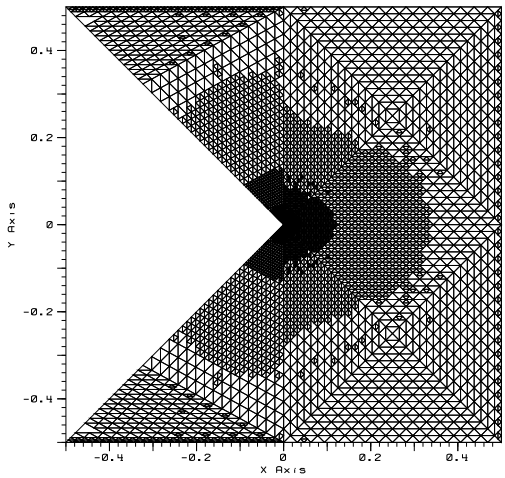
Figure 3.28: Adaptive $||e||_{1,h}$, f6, $r = 2$



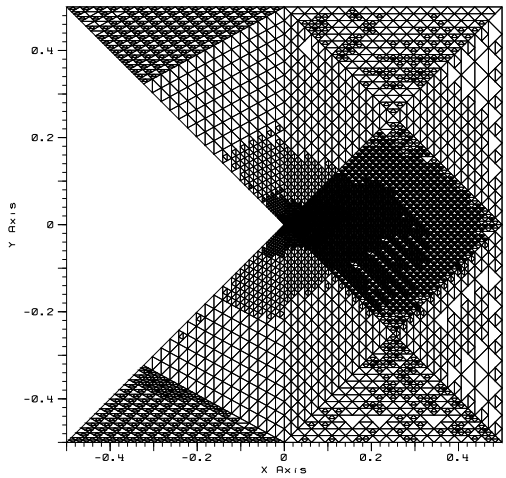
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.29: Adaptive Meshes: f6, r=2

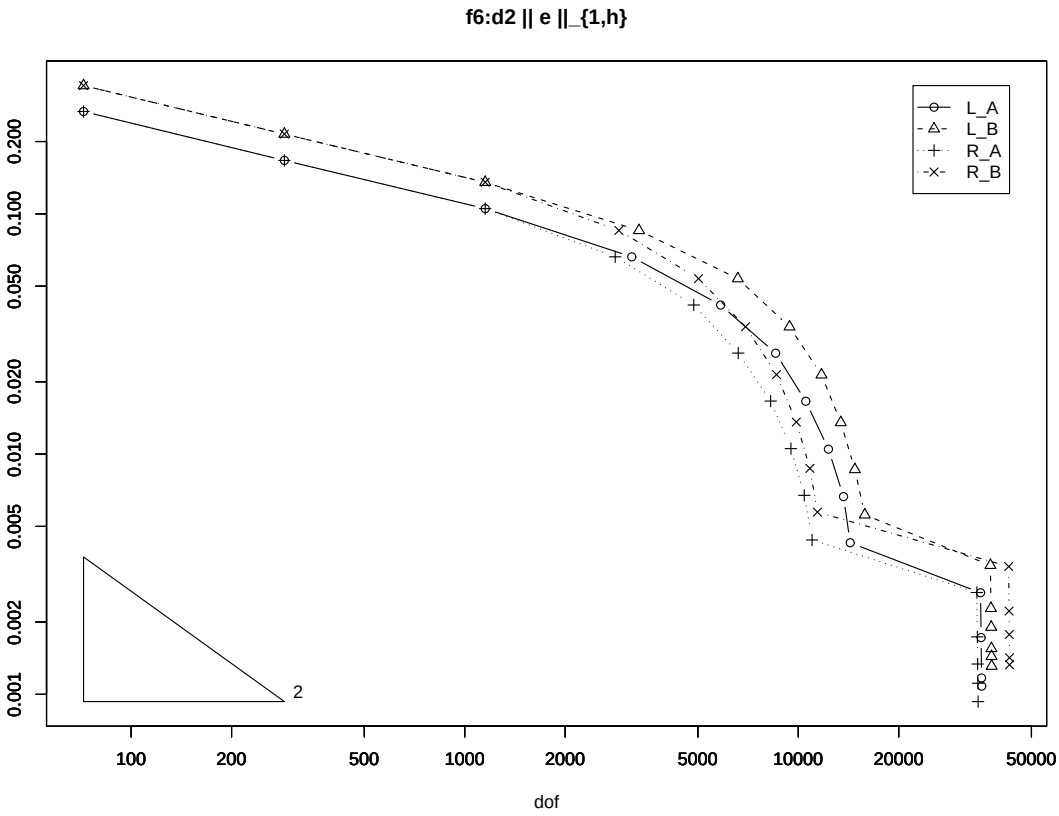
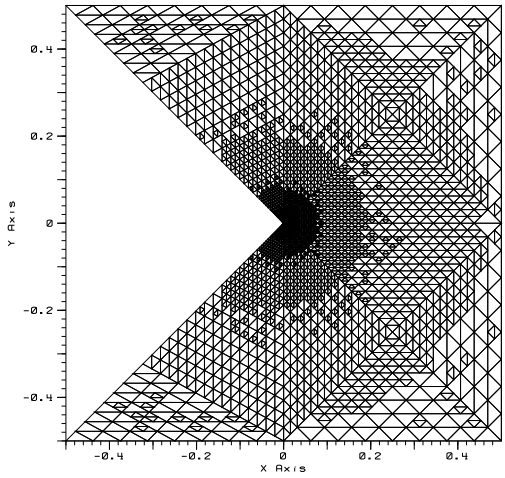
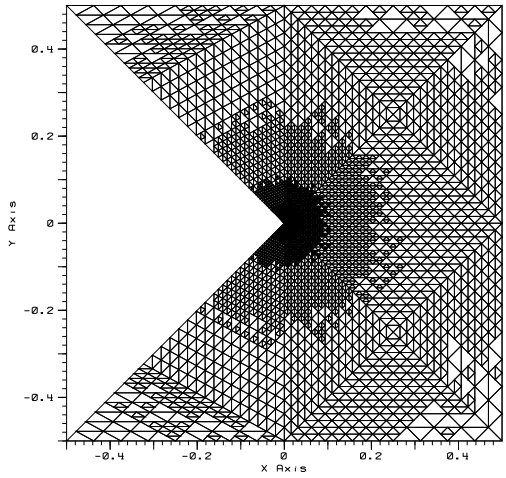


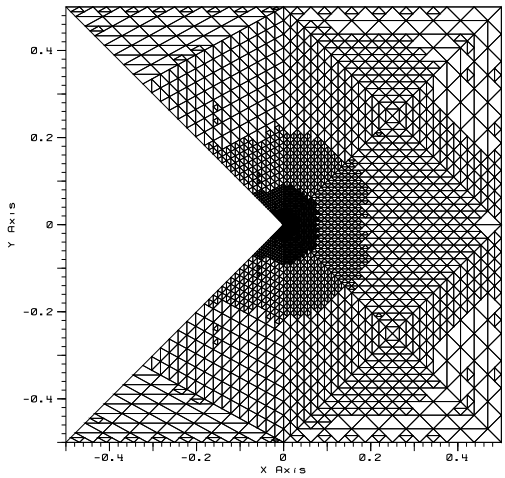
Figure 3.30: Adaptive $||e||_{1,h}$, f6, $r = 3$



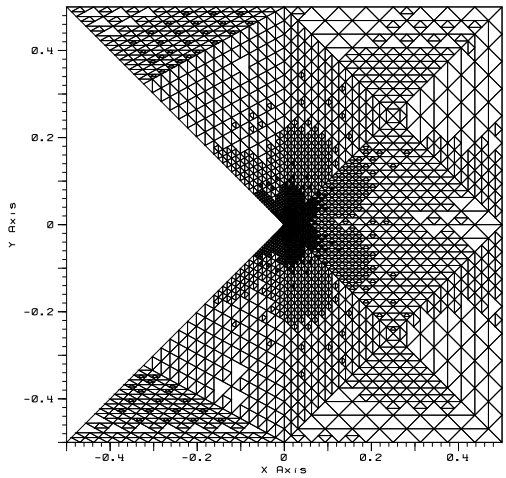
(a) Resid Est., Arnold



(b) Resid Est., Baker



(c) Local Est., Arnold



(d) Local Est., Baker

Figure 3.31: Adaptive Meshes: f6, r=3

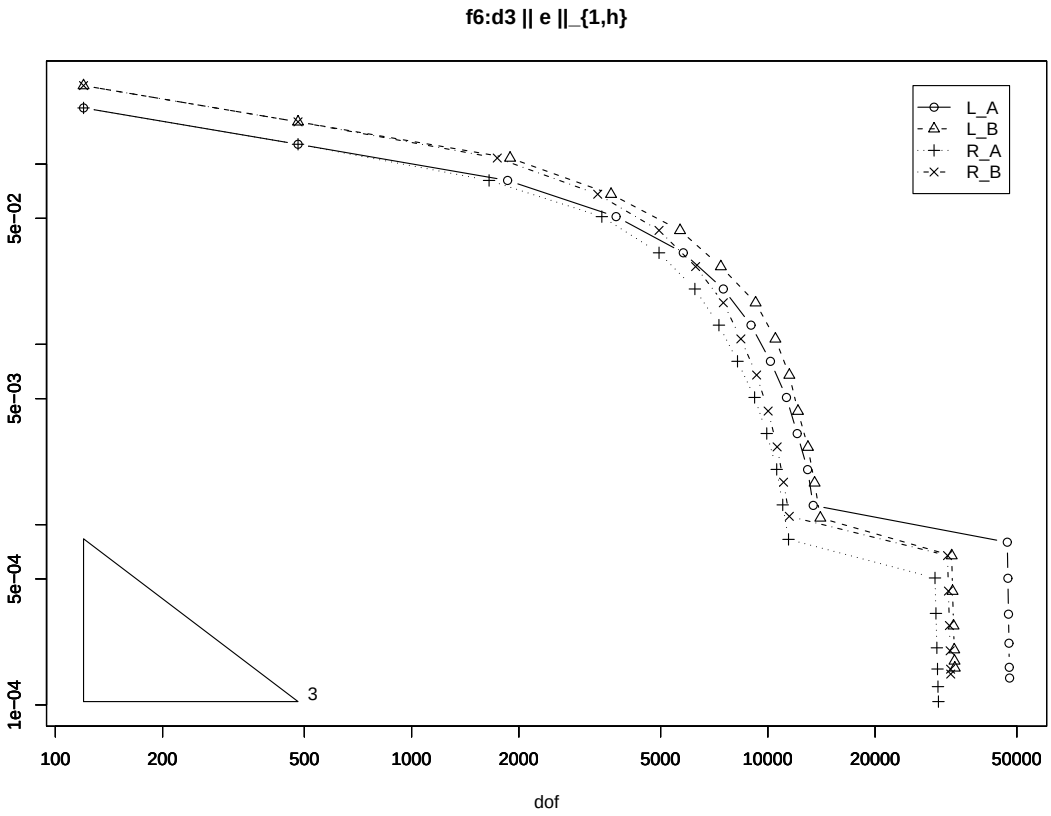
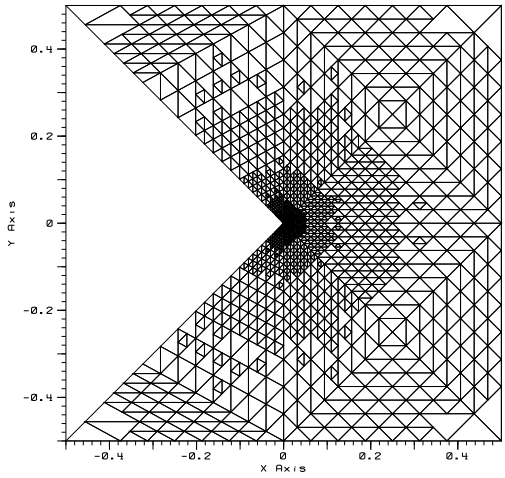
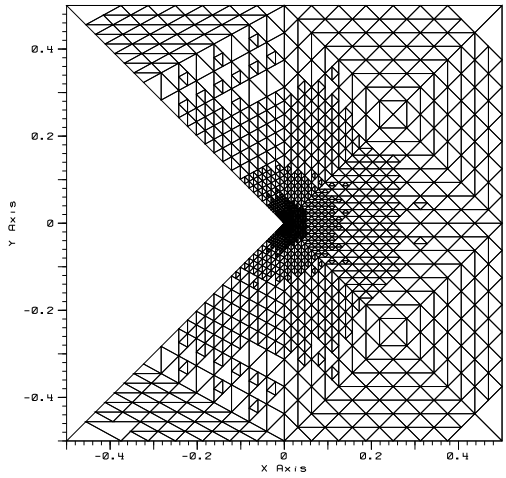


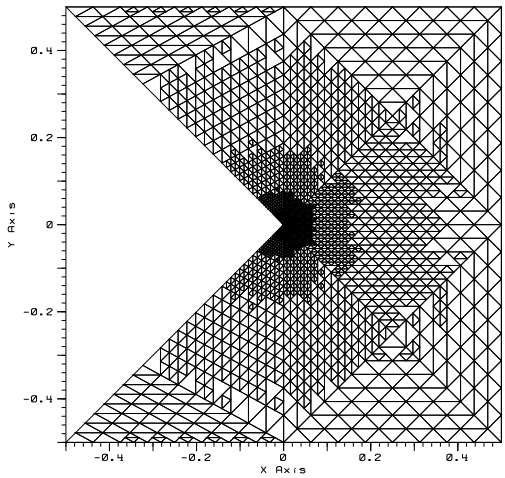
Figure 3.32: Adaptive $||e||_{1,h}$, f6, $r = 4$



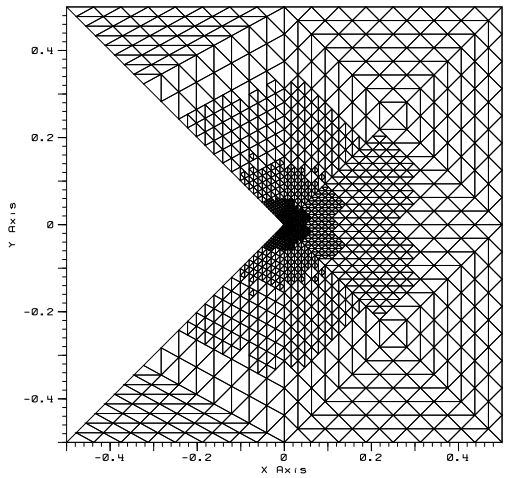
(a) Resid Est., Arnold



(b) Resid Est., Baker

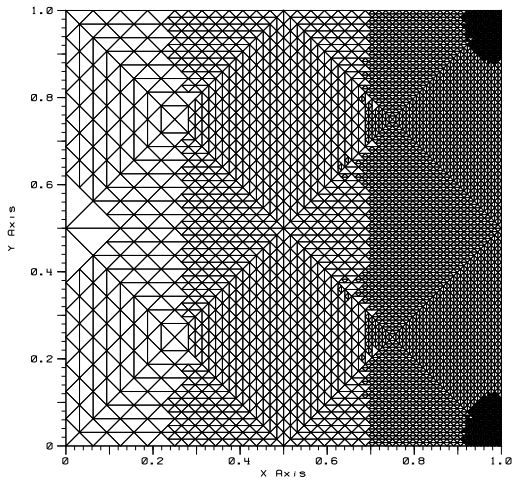


(c) Local Est., Arnold

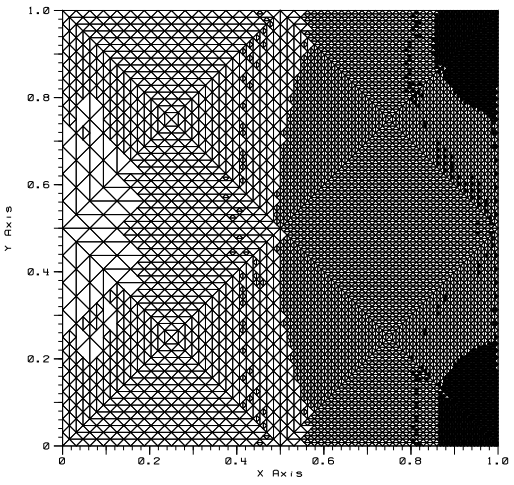


(d) Local Est., Baker

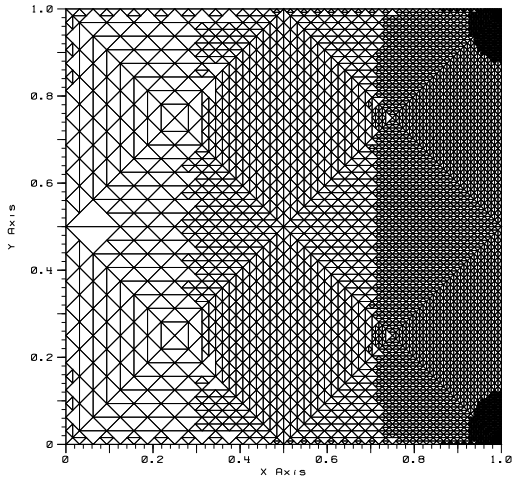
Figure 3.33: Adaptive Meshes: f6, r=4



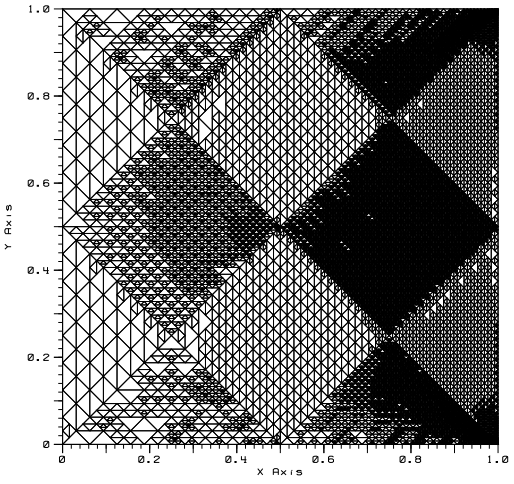
(a) Resid Est., Arnold



(b) Resid Est., Baker

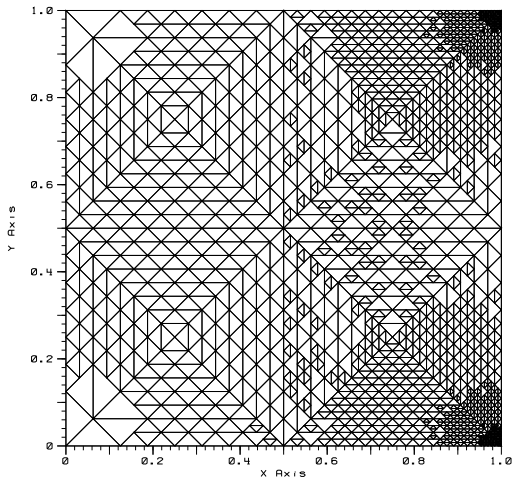


(c) Local Est., Arnold

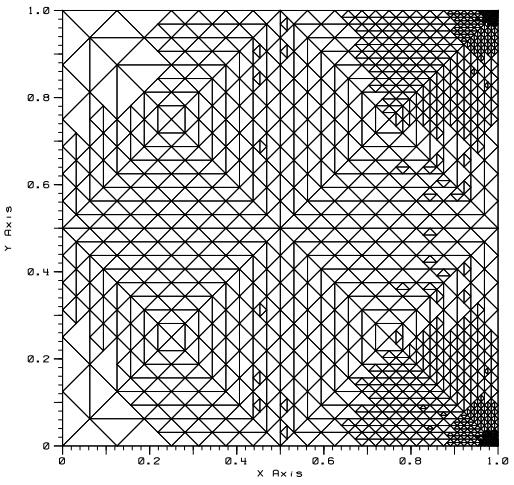


(d) Local Est., Baker

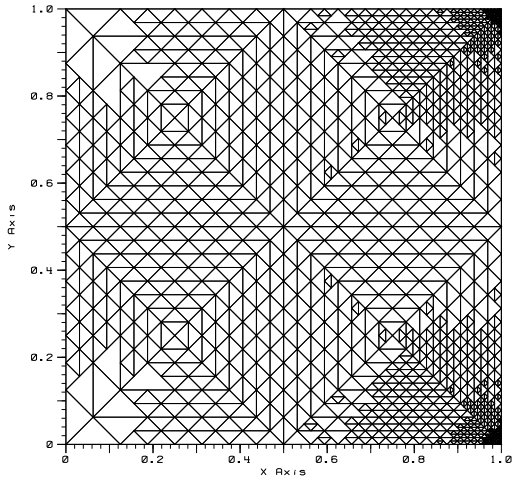
Figure 3.34: Adaptive Meshes: f7, r=2



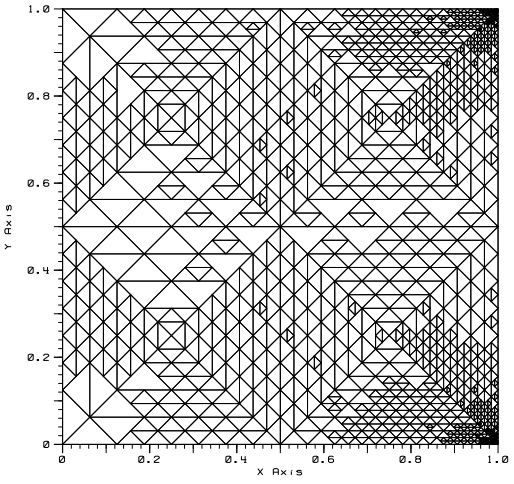
(a) Resid Est., Arnold



(b) Resid Est., Baker

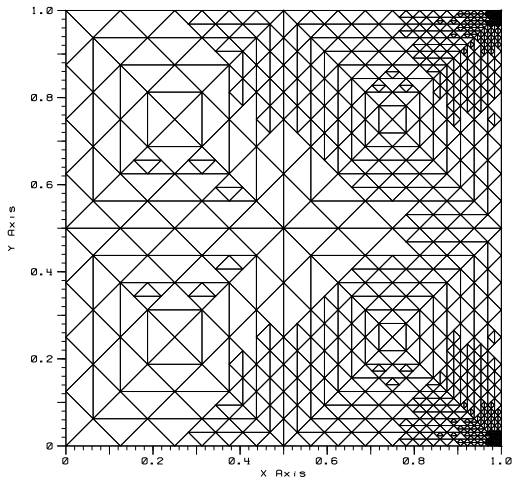


(c) Local Est., Arnold

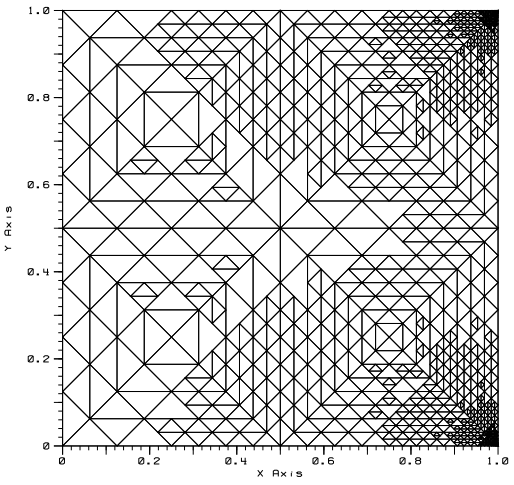


(d) Local Est., Baker

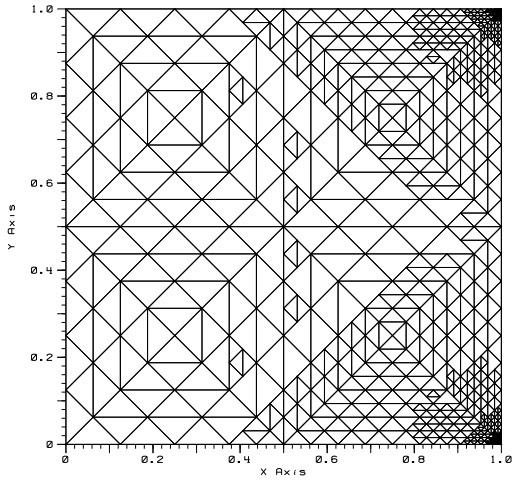
Figure 3.35: Adaptive Meshes: f7, r=3



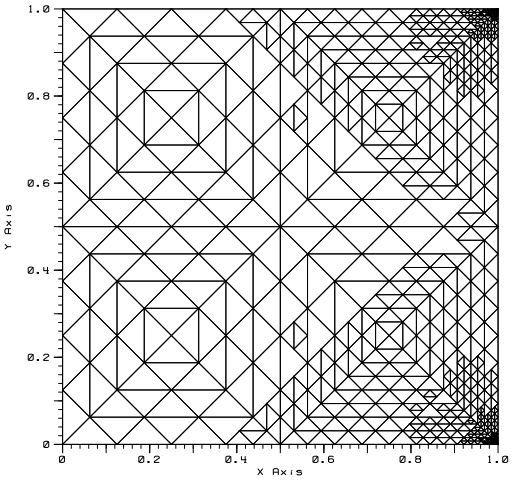
(a) Resid Est., Arnold



(b) Resid Est., Baker

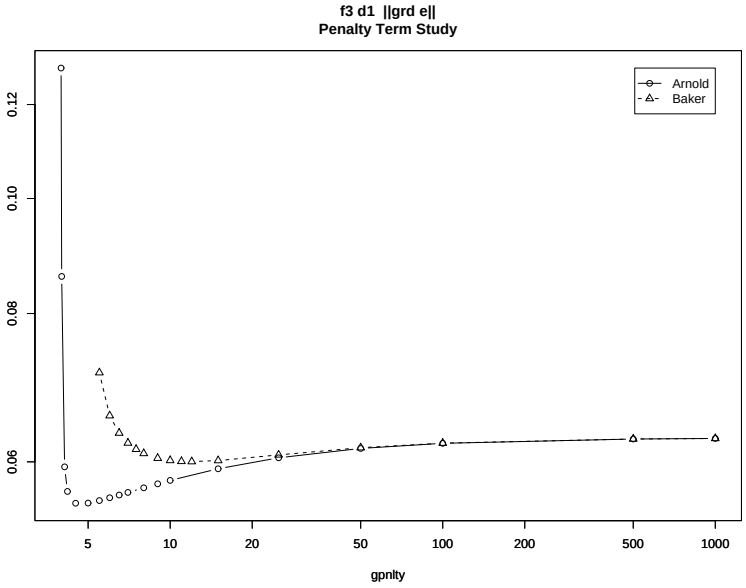


(c) Local Est., Arnold

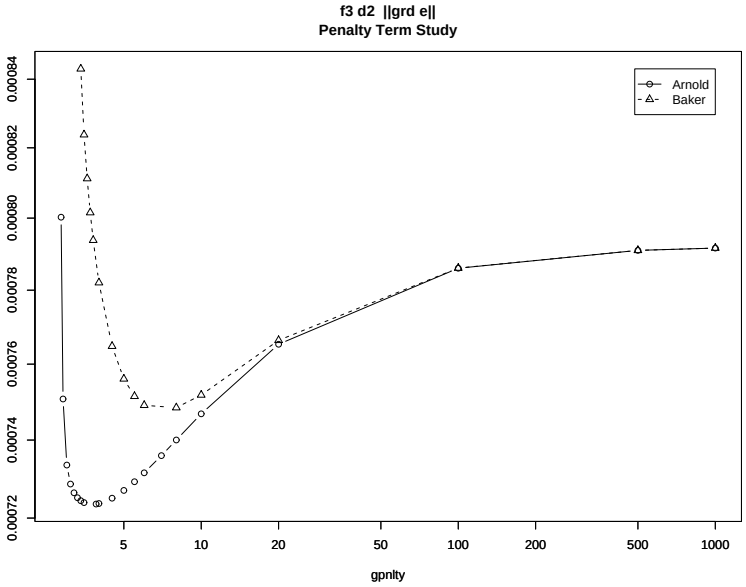


(d) Local Est., Baker

Figure 3.36: Adaptive Meshes: f7, r=4

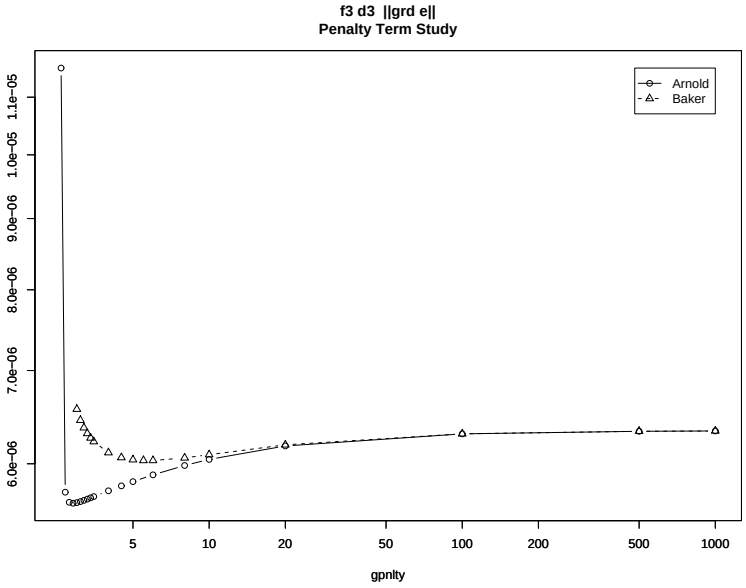


(a) f3, $r = 2$

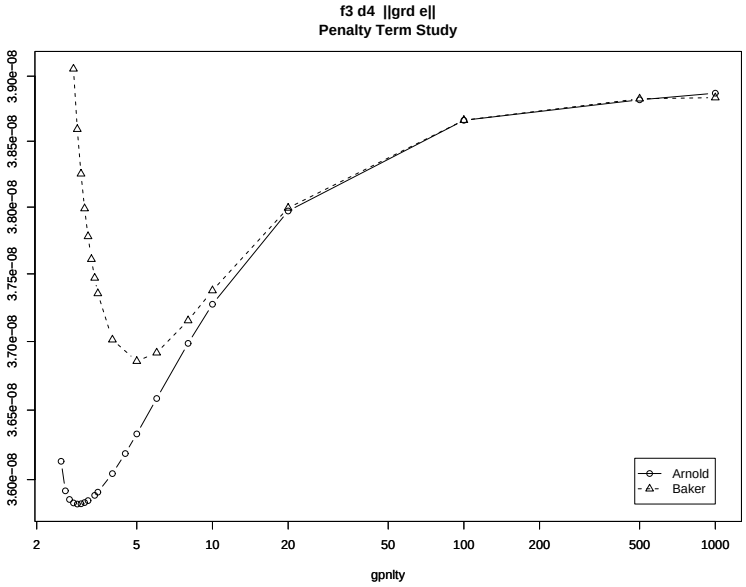


(b) f3, $r = 3$

Figure 3.37: γ Study, $\|\nabla e\|$, f3, $r = 2, 3$

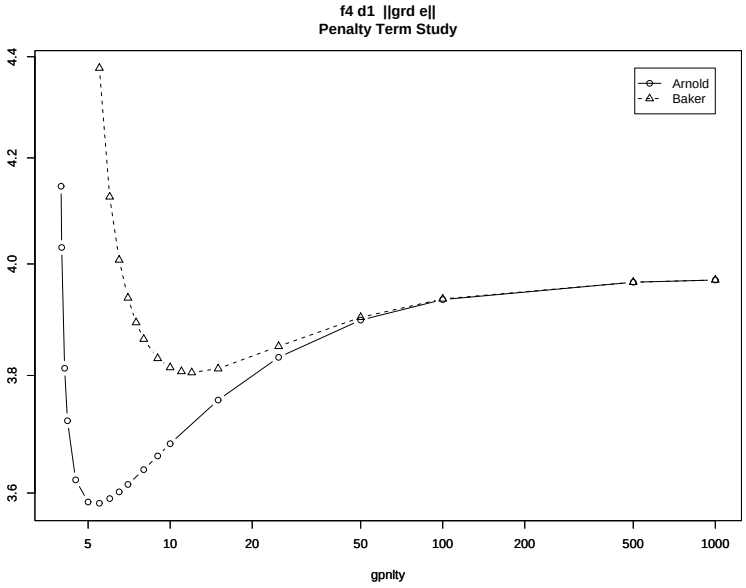


(a) f3, $r = 4$

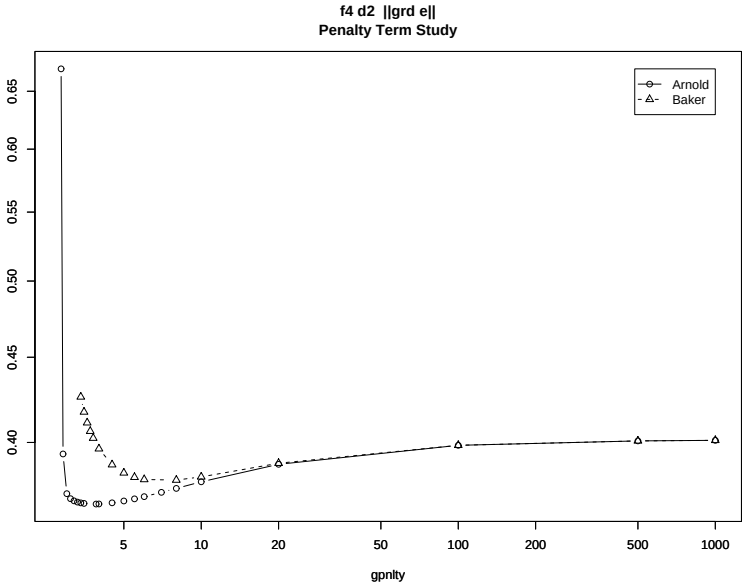


(b) f3, $r = 5$

Figure 3.38: γ Study, $\|\nabla e\|$, f3, $r = 4, 5$

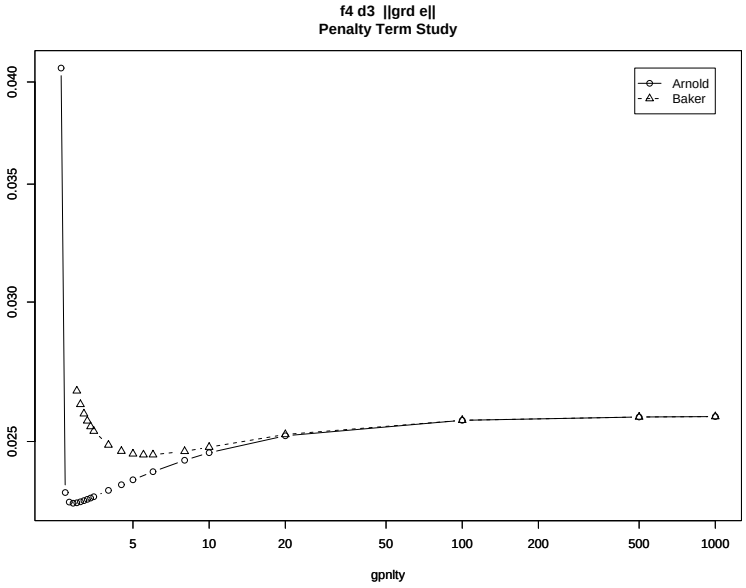


(a) f4, $r = 2$

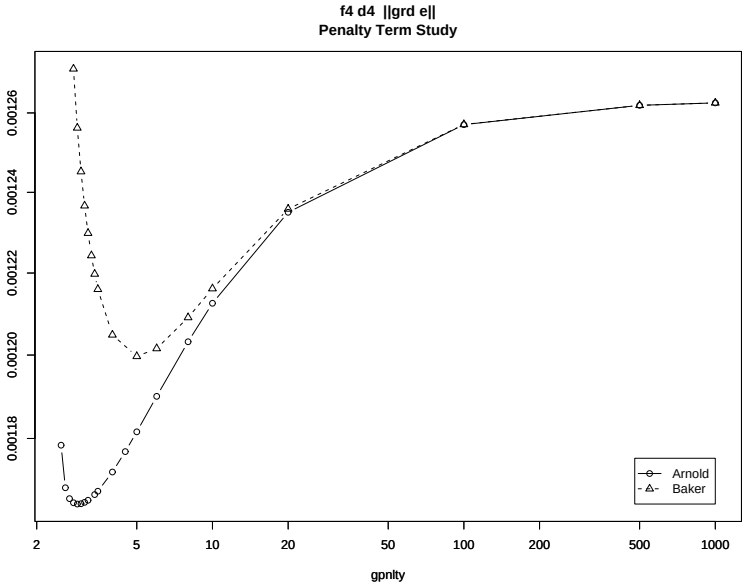


(b) f4, $r = 3$

Figure 3.39: γ Study, $\|\nabla e\|$, f4, $r = 2, 3$

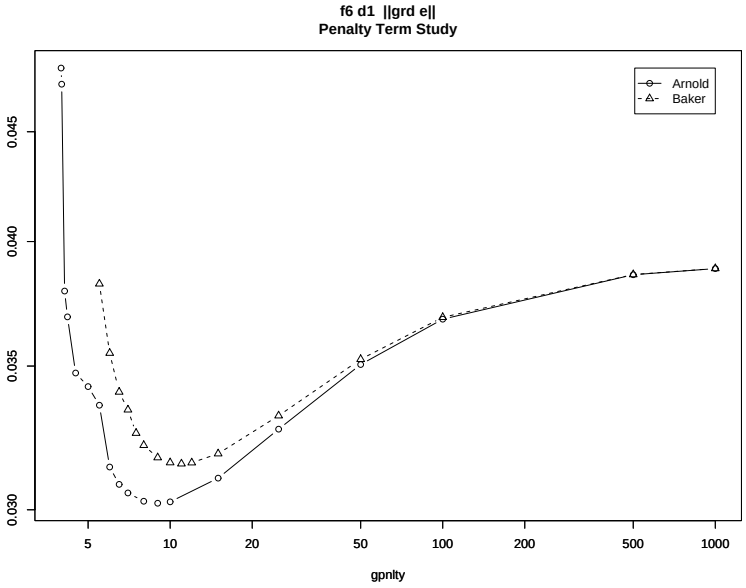


(a) f4, $r = 4$

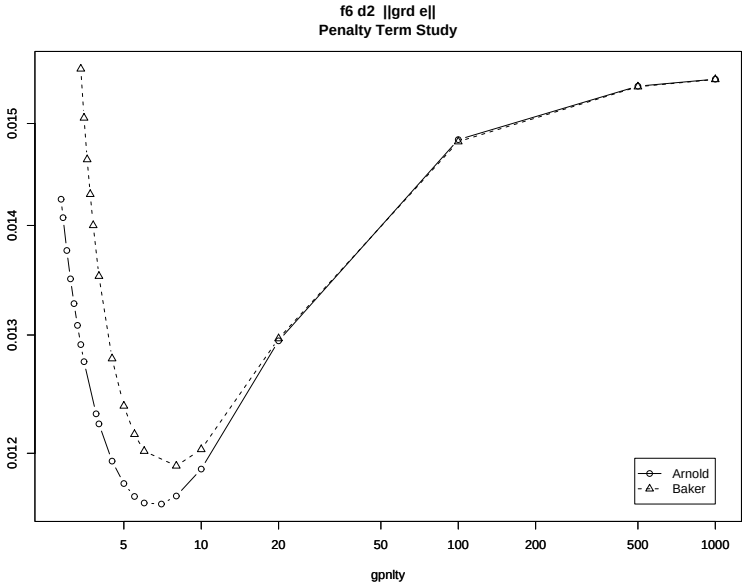


(b) f4, $r = 5$

Figure 3.40: γ Study, $\|\nabla e\|$, f4, $r = 4, 5$

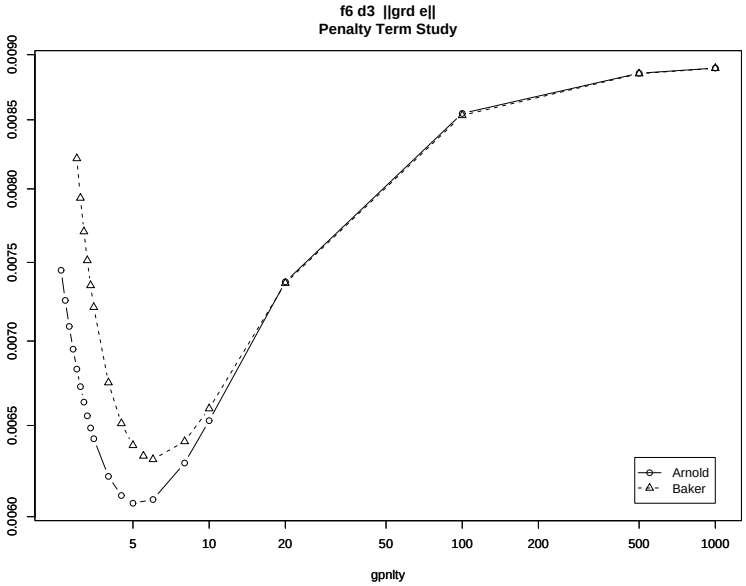


(a) f6, $r = 2$

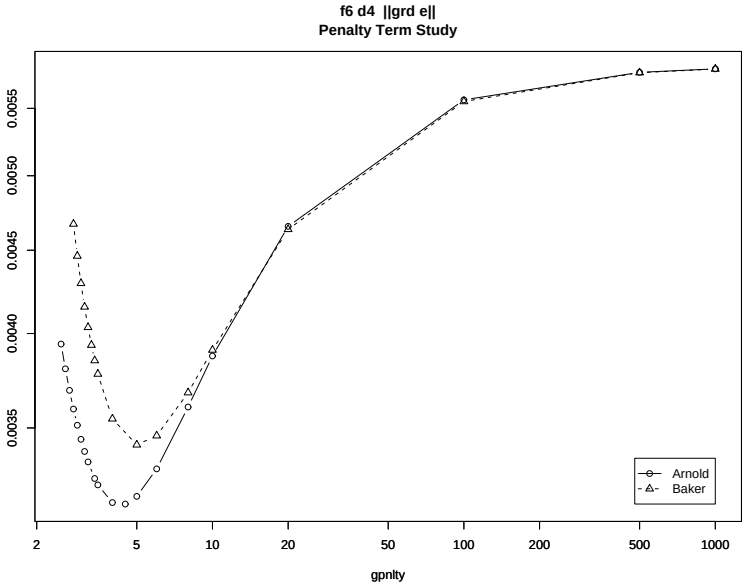


(b) f6, $r = 3$

Figure 3.41: γ Study, $\|\nabla e\|$, f6, $r = 2, 3$

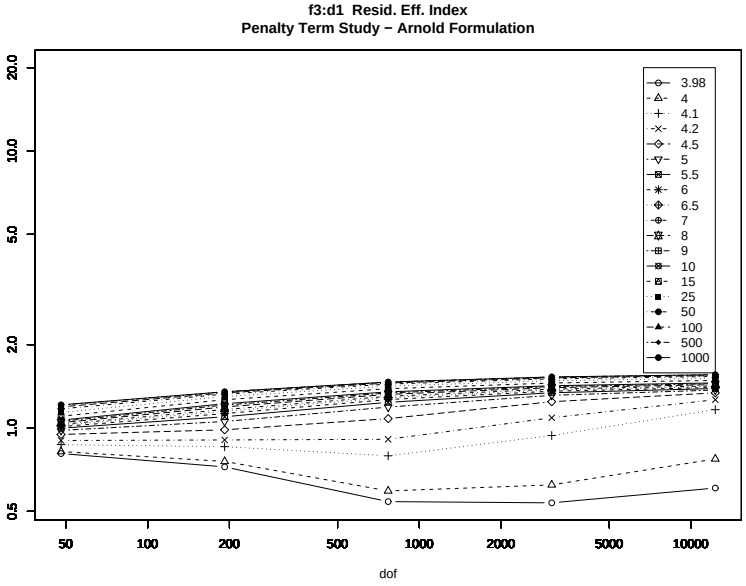


(a) f6, $r = 4$

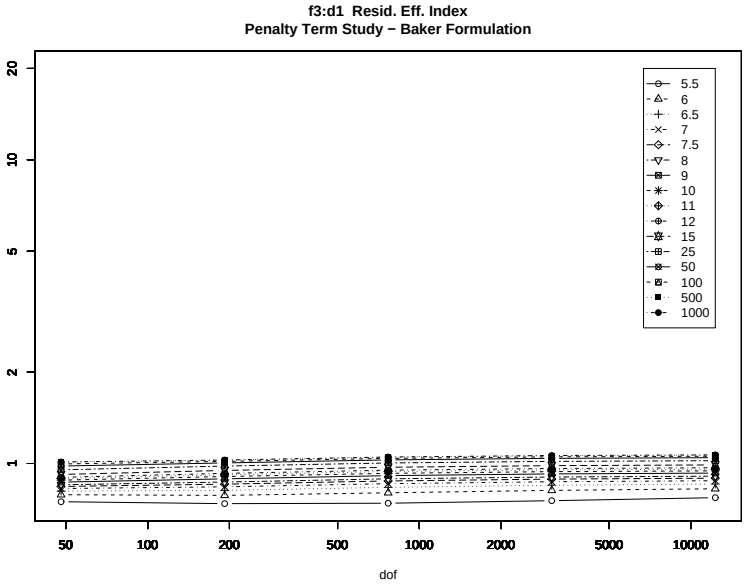


(b) f6, $r = 5$

Figure 3.42: γ Study, $\|\nabla e\|$, f6, $r = 4, 5$

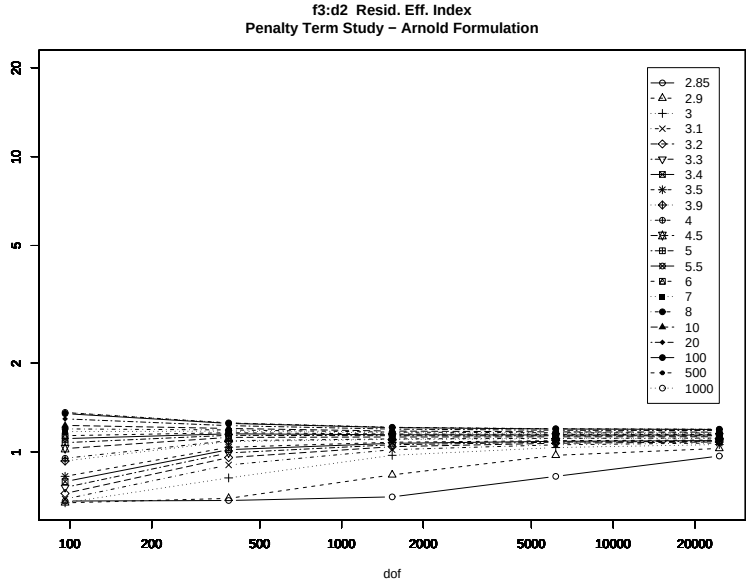


(a) f3, Arnold, Residual

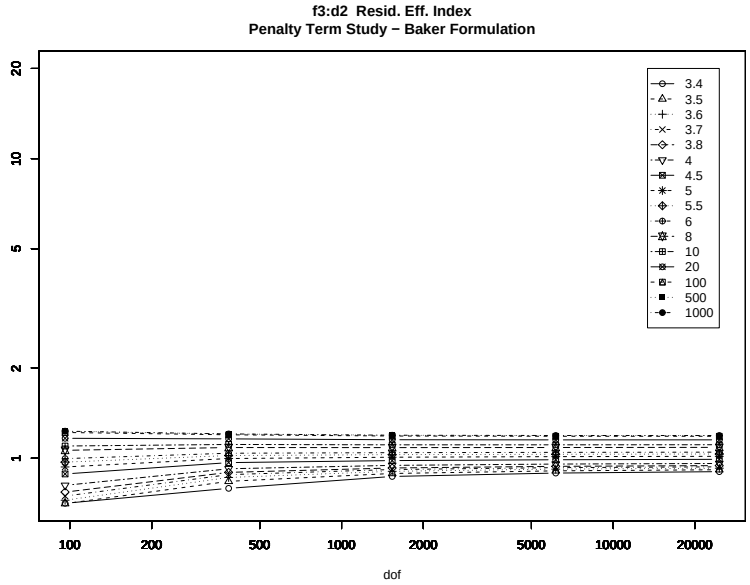


(b) f3, Baker, Residual

Figure 3.43: γ Study, Effectivity Indices, f3, $r = 2$

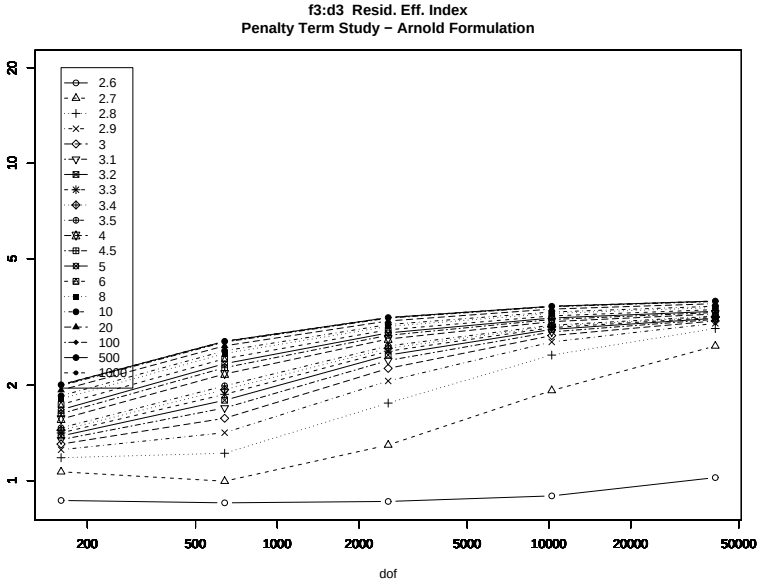


(a) f3, Arnold, Residual

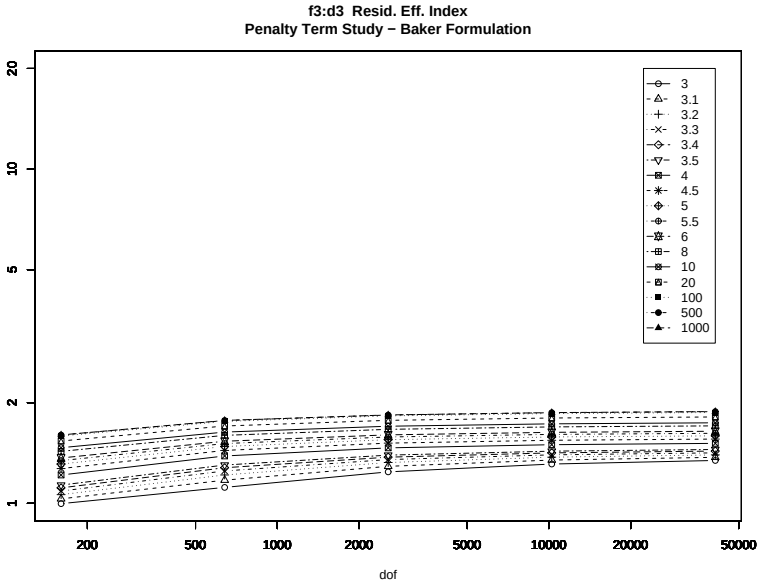


(b) f3, Baker, Residual

Figure 3.44: γ Study, Effectivity Indices, f3, $r = 3$

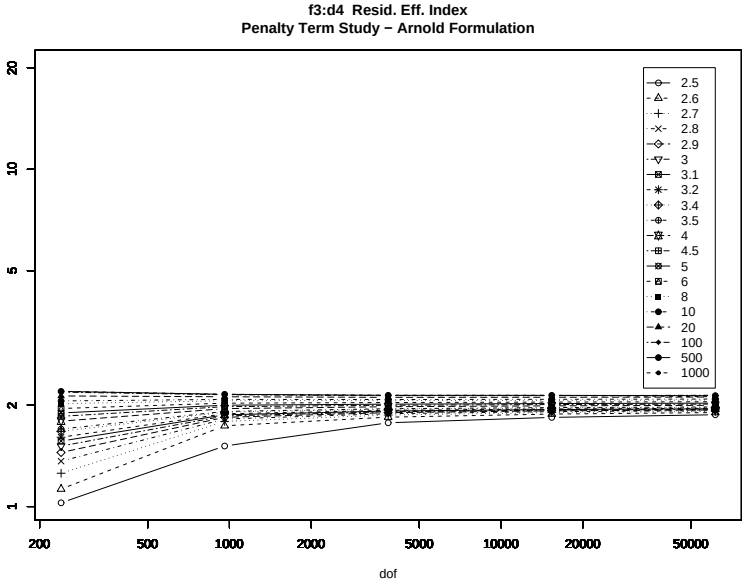


(a) f3, Arnold, Residual

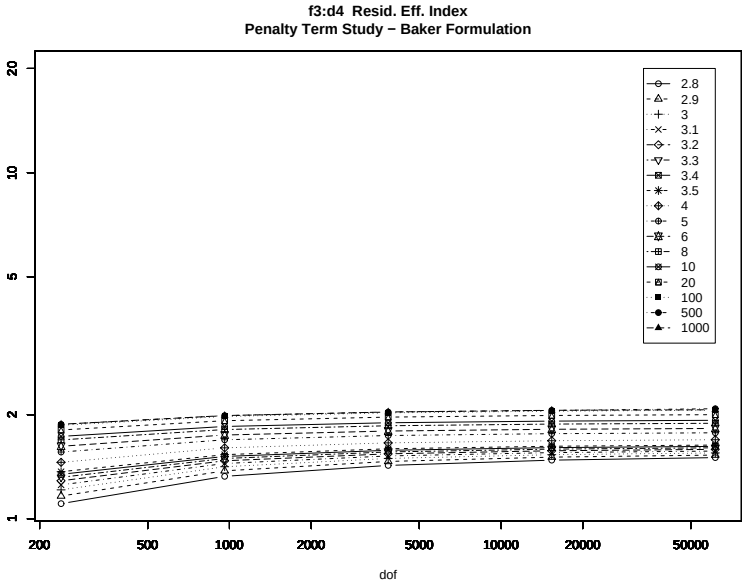


(b) f3, Baker, Residual

Figure 3.45: γ Study, Effectivity Indices, f3, $r = 4$

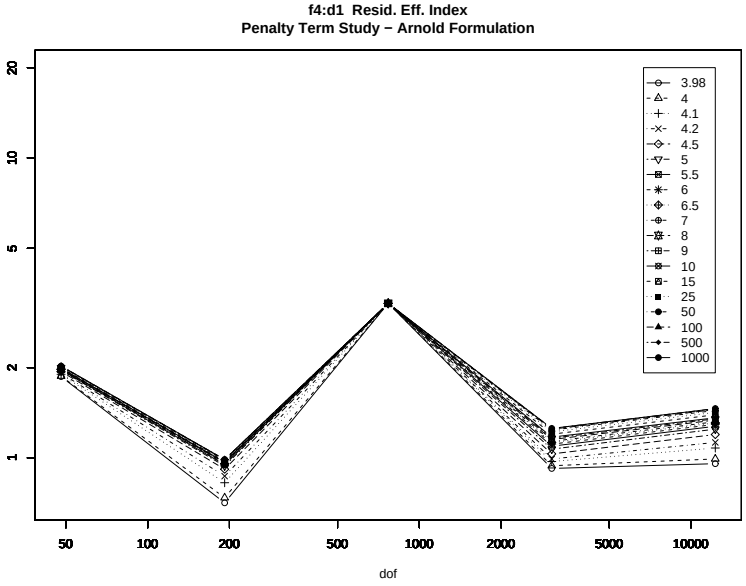


(a) f3, Arnold, Residual

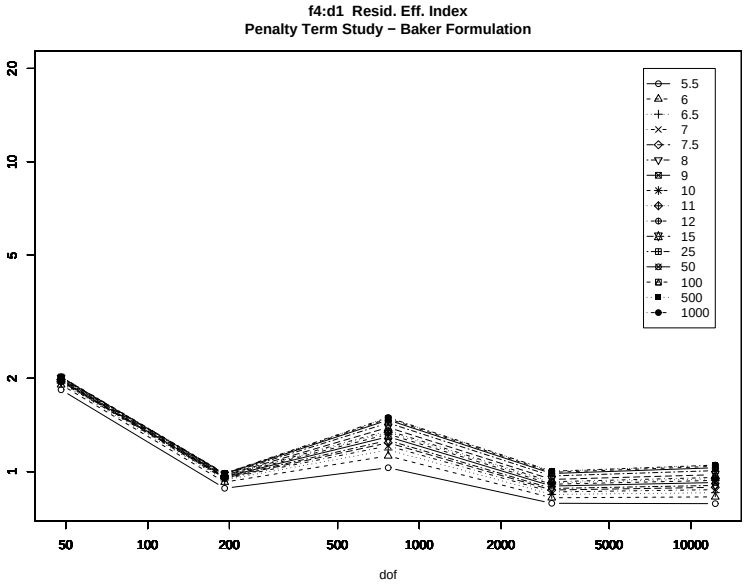


(b) f3, Baker, Residual

Figure 3.46: γ Study, Effectivity Indices, f3, $r = 5$

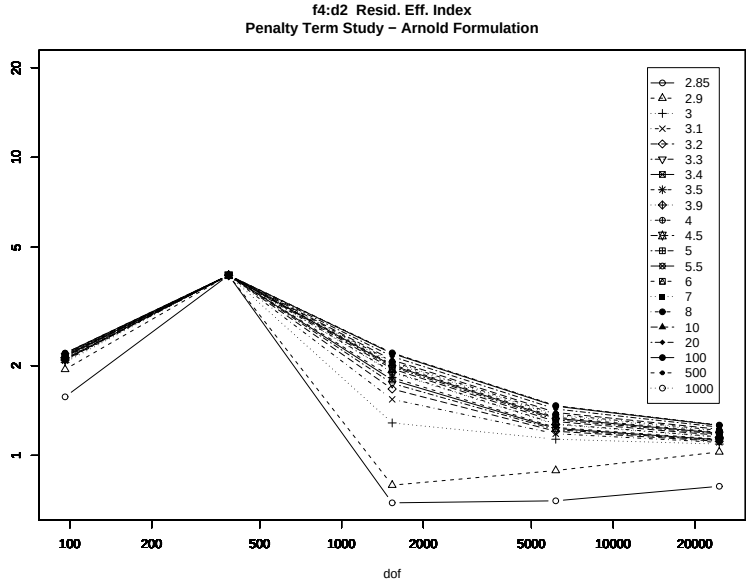


(a) f4, Arnold, Residual

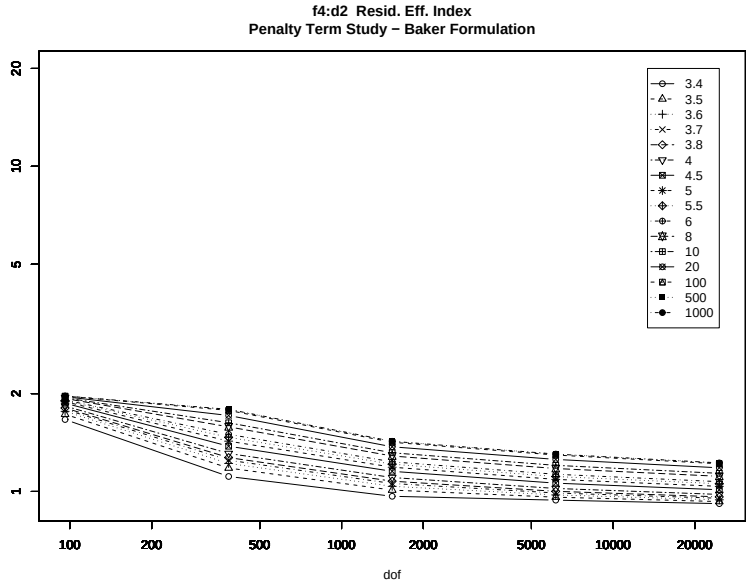


(b) f3, Baker, Residual

Figure 3.47: γ Study, Effectivity Indices, f4, $r = 2$

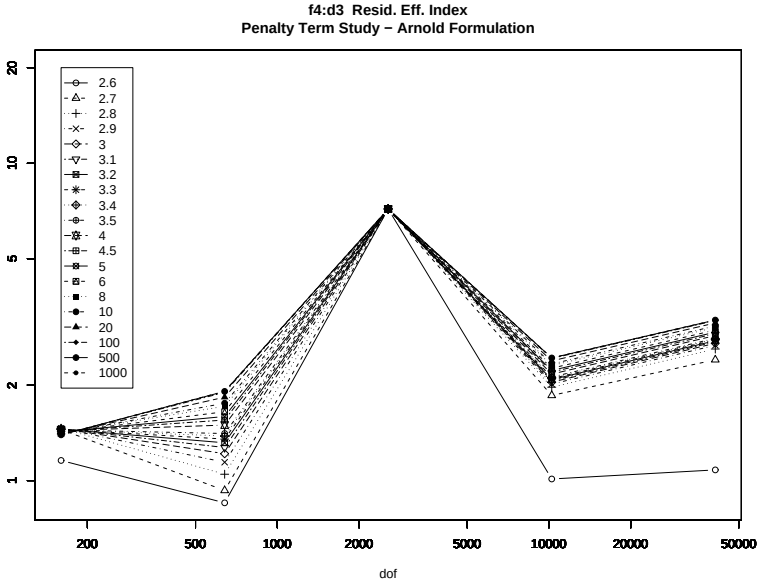


(a) f4, Arnold, Residual

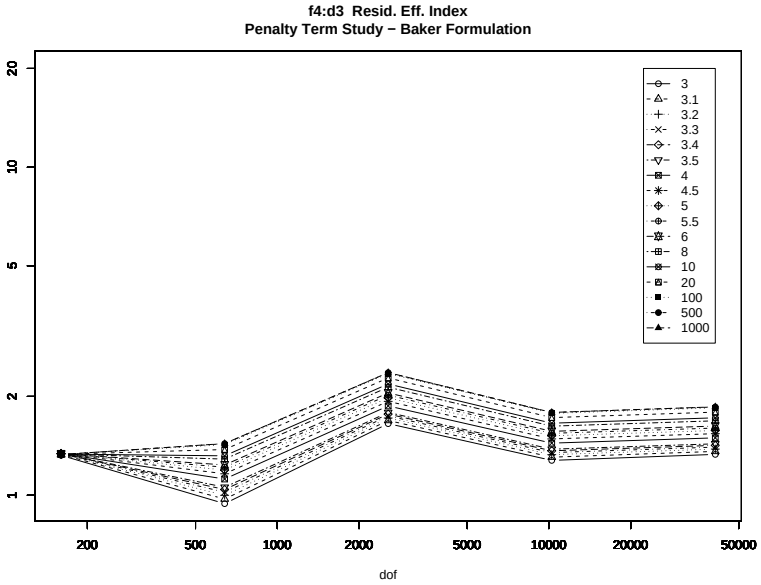


(b) f4, Baker, Residual

Figure 3.48: γ Study, Effectivity Indices, f4, $r = 3$

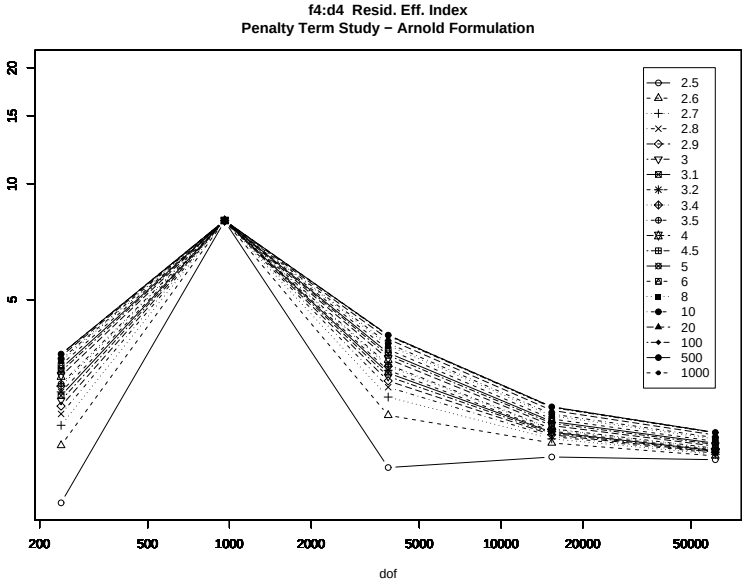


(a) f4, Arnold, Residual

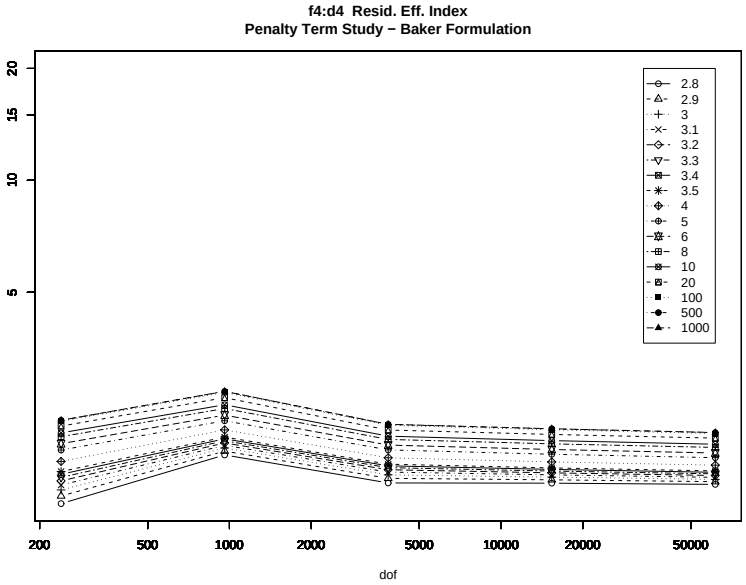


(b) f4, Baker, Residual

Figure 3.49: γ Study, Effectivity Indices, f4, $r = 4$

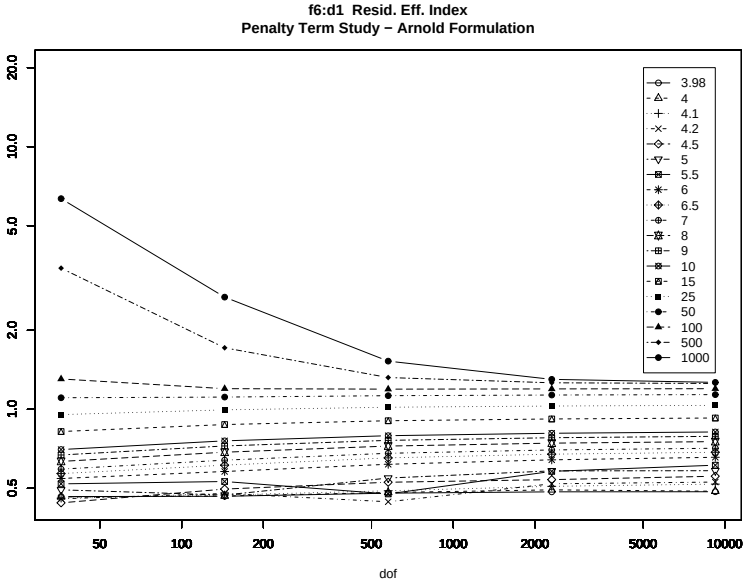


(a) f4, Arnold, Residual

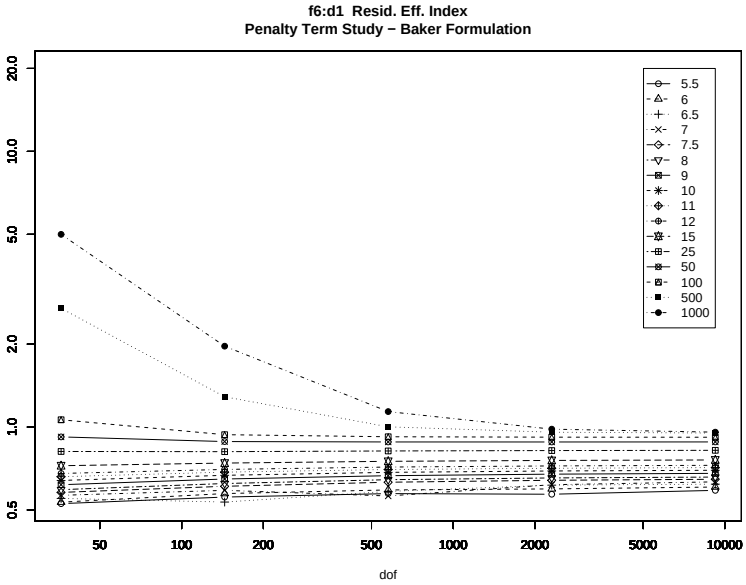


(b) f4, Baker, Residual

Figure 3.50: γ Study, Effectivity Indices, f4, $r = 5$

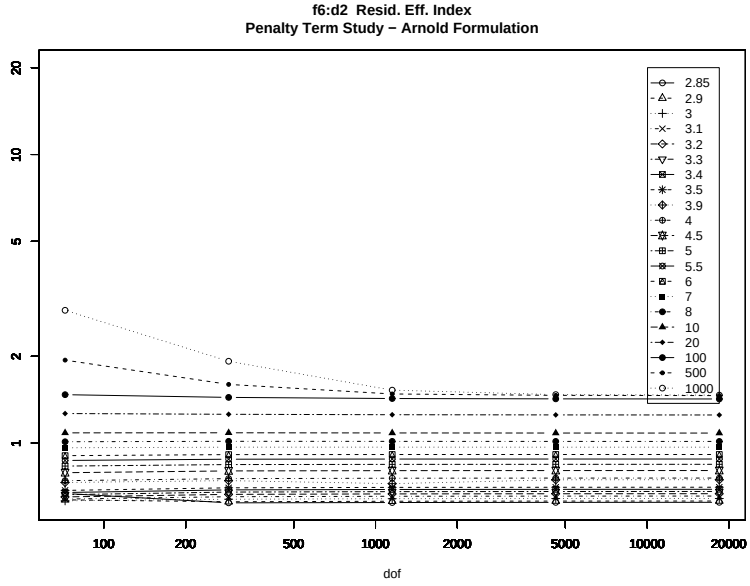


(a) f6, Arnold, Residual

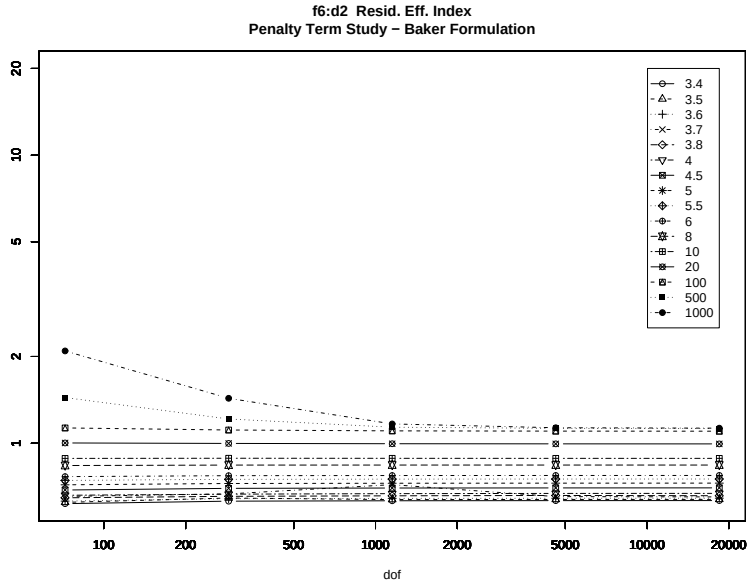


(b) f6, Baker, Residual

Figure 3.51: γ Study, Effectivity Indices, f6, $r = 2$

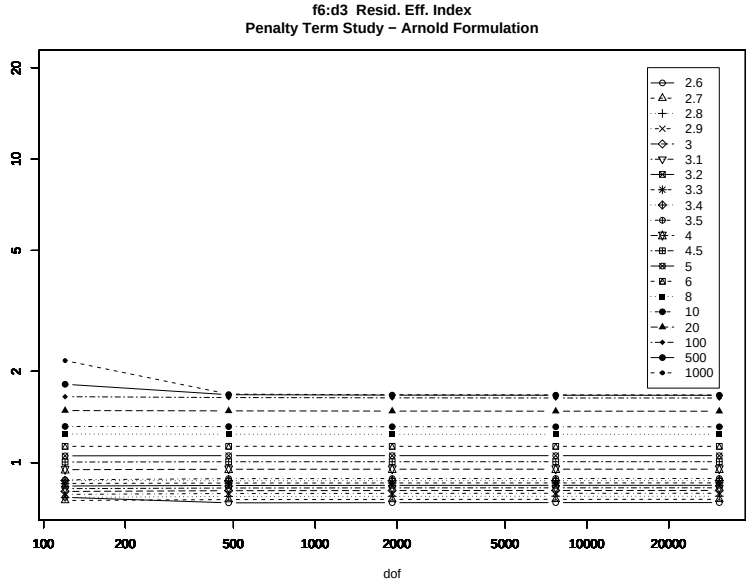


(a) f6, Arnold, Residual

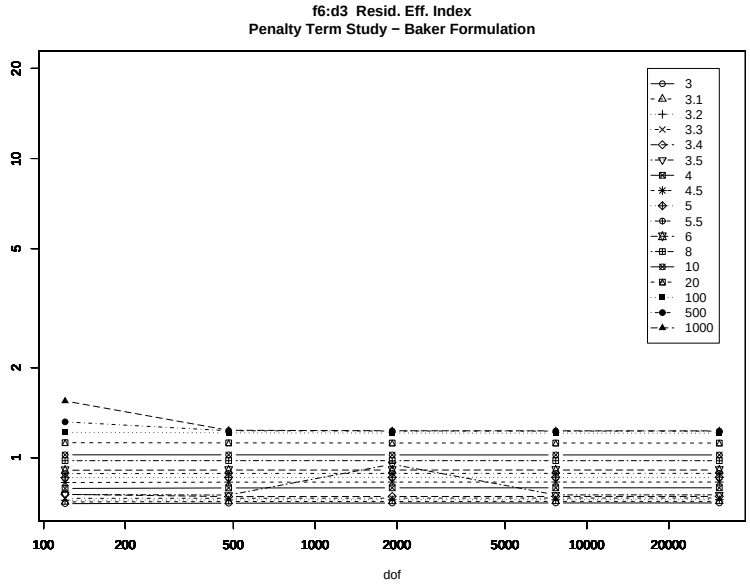


(b) f6, Baker, Residual

Figure 3.52: γ Study, Effectivity Indices, f6, $r = 3$

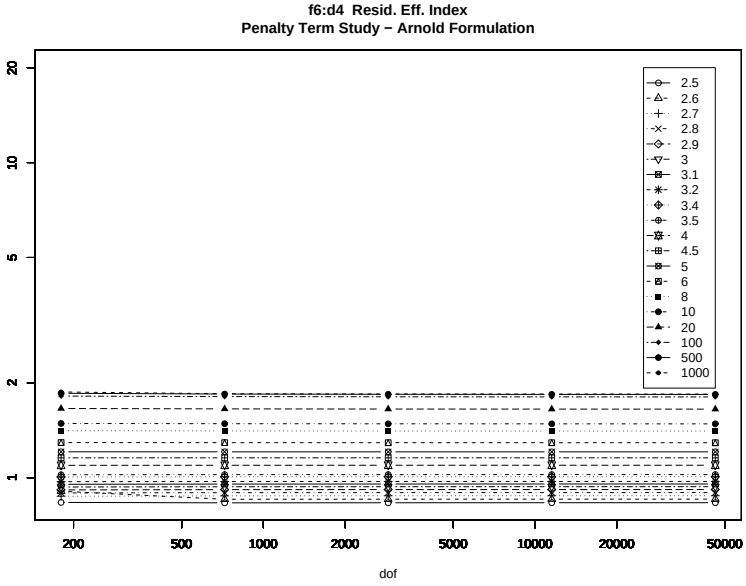


(a) f6, Arnold, Residual

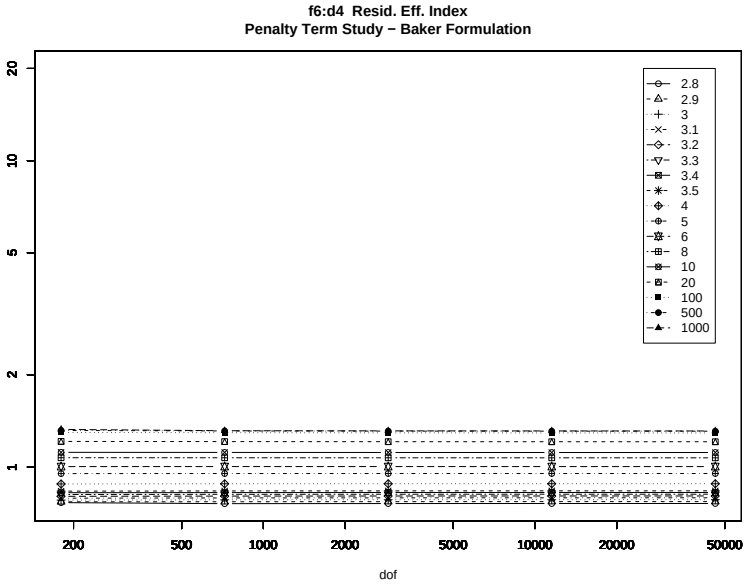


(b) f6, Baker, Residual

Figure 3.53: γ Study, Effectivity Indices, f6, $r = 4$



(a) f6, Arnold, Residual



(b) f6, Baker, Residual

Figure 3.54: γ Study, Effectivity Indices, f6, $r = 5$

Chapter 4

Biharmonic Problems

4.1 DG Formulation

4.1.1 Model Problem

The fourth order elliptic model problem under consideration is:

$$\begin{cases} \Delta^2 u = f & \text{in } \Omega \\ u = g_D & \text{on } \Gamma \\ \nabla u \cdot n = g_N & \text{on } \Gamma \end{cases} \tag{MP}$$

where $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ and $\partial\Omega = \Gamma$ with n being the unit outward normal vector to Γ.

Let the *energy spaces* E_h be defined as

$$E_h = \prod_{K \in \mathcal{T}_h} H^4(K)$$

and the *finite element spaces* V_h^r be defined as

$$V_h^r = \prod_{K \in \mathcal{T}_h} P_r(K)$$

where $P_r(K)$ denotes the space of polynomials of total degree $r-1$. Note that $V_h^r \subset E_h \subset L^2(\Omega)$, but $V_h^r \not\subset H^2(\Omega)$ and $V_h^r \not\subset H^1(\Omega)$.

4.1.2 SIPG Formulation

After working through the details (to be included in dissertation), the SIPG problem is:

Problem (Fourth Order Elliptic Weak Formulation). *Find* $u_h^\gamma \in V_h^r$ *such that*

$$a_h^\gamma(u_h^\gamma, v) = F_h^\gamma(v), \qquad \forall v \in V_h^r \tag{4.1}$$

where

$$a_h^\gamma\left(u_h^\gamma, v\right) = \sum_{K\in\mathcal{T}_h} \left(\Delta u_h^\gamma, \Delta v\right)_K + \sum_{e\in\mathcal{E}} \Bigg(\left\langle \{\partial_n(\Delta v)\}, [u_h^\gamma] \right\rangle_e - \left\langle \{\Delta v\}, [\partial_n u_h^\gamma] \right\rangle_e + \left\langle \{\partial_n(\Delta u_h^\gamma)\}, [v] \right\rangle_e - \left\langle \{\Delta u_h^\gamma\}, [\partial_n v] \right\rangle_e + \gamma h_e^{-1} \left\langle [\partial_n u_h^\gamma], [\partial_n v] \right\rangle_e + \gamma h_e^{-3} \left\langle [u_h^\gamma], [v] \right\rangle_e \Bigg) \tag{4.2}$$

and

$$F_h^\gamma(v) = \sum_{K\in\mathcal{T}_h} (f, v)_K + \sum_{e\in\Gamma} \left(\left\langle g_D, \partial_n(\Delta v) + \gamma h_e^{-3} v \right\rangle_e + \left\langle g_N, \gamma h_e^{-1} \partial_n v - \Delta v \right\rangle_e \right) \tag{4.3}$$

The bilinear form $a_h^\gamma(\cdot,\cdot)$ induces the following norms on E_h:

$$\|v\|_{2,h} = \left(\sum_{K\in\mathcal{T}_h} \|\Delta v\|_{0,K}^2 + \sum_{e\in\mathcal{E}} \left(h_e^{-3} |[v]|_{0,e}^2 + h_e^{-1} \left|[\partial_n v]\right|_{0,e}^2 + h_e |\{\Delta v\}|_{0,e}^2 + h_e^3 \left|\{\partial_n(\Delta v)\}\right|_{0,e}^2 \right) \right)^{1/2} \tag{4.4}$$

and

$$\|v\|_{1,h} = \left(\sum_{K\in\mathcal{T}_h} \|\nabla v\|_{0,K}^2 + \sum_{e\in\mathcal{E}} \left(h_e^{-1} |[v]|_{0,e}^2 + h_e \left|\{\partial_n v\}\right|_{0,e}^2 \right) \right)^{1/2} \tag{4.5}$$

There are two different forms for $\{\partial_n v\}\big|_e$:

1. **Arnold**: $\{\partial_n v\}\big|_e = \dfrac{1}{2}\left(\dfrac{\partial v^+}{\partial n^+} + \dfrac{\partial v^-}{\partial n^+} \right)\Bigg|_e$
2. **Baker**: $\{\partial_n v\}\big|_e = \dfrac{\partial v^+}{\partial n^+}\Bigg|_e$

and of course $[v]\big|_e = v^+\big|_e - v^-\big|_e$.

4.1.3 Stiffness Matrix Assembly

Assembly of the stiffness matrix A requires subassembly of the diagonal blocks A_{ii} and the off-diagonal blocks A_{ij}, $i,j = 1,2,\ldots,|\mathcal{T}_h|$. Note that $A_{ii} = A_{ii}^\mathsf{T}$ and $A_{ij} = A_{ji}^\mathsf{T}$ implies that $A = A^\mathsf{T}$. Thus, assembly of A requires subassembly of $A_K, \forall K \in \mathcal{T}_h$ and $A_e, \forall e \in \mathcal{E}^I$. Note here that since each interior edge e is the common boundary between two triangles K^+, K^-, A_e is the off-diagonal block describing interaction between degrees of freedom of K^+ and degrees of freedom of K^- through edge e.

The first step is to rewrite Eq (4.2) into a form where individual components can be clearly

identified as being associated with either A_K or A_e. Expanding the jump terms gives:

$$
\begin{aligned}
a_h^\gamma(u,v) = \sum_{K\in\mathcal{T}_h} (\Delta u, \Delta v)_K + \sum_{e\in\mathcal{E}} \Bigg(& \left\langle \{\partial_n(\Delta v)\}, u^+ \right\rangle_e - \left\langle \{\Delta v\}, \partial_n u^+ \right\rangle_e + \left\langle \{\partial_n(\Delta u)\}, v^+ \right\rangle_e \\
& - \left\langle \{\Delta u\}, \partial_n v^+ \right\rangle_e + \gamma h_e^{-1} \left\langle \partial_n u^+, \partial_n v^+ \right\rangle_e + \gamma h_e^{-3} \left\langle u^+, v^+ \right\rangle_e \Bigg) \\
- \sum_{e\in\mathcal{E}^I} \Bigg(& \left\langle \{\partial_n(\Delta v)\}, u^- \right\rangle_e - \left\langle \{\Delta v\}, \partial_n u^- \right\rangle_e + \left\langle \{\partial_n(\Delta u)\}, v^- \right\rangle_e - \left\langle \{\Delta u\}, \partial_n v^- \right\rangle_e \\
& + \gamma h_e^{-1} \left\langle \partial_n u^-, \partial_n v^+ \right\rangle_e + \gamma h_e^{-3} \left\langle u^-, v^+ \right\rangle_e + \gamma h_e^{-1} \left\langle \partial_n u^+, \partial_n v^- \right\rangle_e + \gamma h_e^{-3} \left\langle u^+, v^- \right\rangle_e \Bigg) \\
& + \sum_{e\in\mathcal{E}^I} \left(\gamma h_e^{-1} \left\langle \partial_n u^-, \partial_n v^- \right\rangle_e + \gamma h_e^{-3} \left\langle u^-, v^- \right\rangle_e \right)
\end{aligned} \tag{4.6}
$$

A couple of notes regarding Eq (4.6). First, any terms involving both u^+, v^+ or u^-, v^- are part of A_{K^+} or A_{K^-}, respectively. Second, any terms involving u^+, v^- or u^-, v^+ are a part of A_e. Third, Eq (4.6) takes different forms depending on whether the Arnold or Baker forms of $\{\partial_n u\}$ and $\{\partial_n v\}$ are chosen. For the Baker formulation, Eq (4.6) becomes

$$
\begin{aligned}
a_h^\gamma(u,v) = \sum_{K\in\mathcal{T}_h} (\Delta u, \Delta v)_K + \sum_{e\in\mathcal{E}} \Bigg(& \left\langle \partial_n(\Delta v^+), u^+ \right\rangle_e - \left\langle \Delta v^+, \partial_n u^+ \right\rangle_e + \left\langle \partial_n(\Delta u^+), v^+ \right\rangle_e \\
& - \left\langle \Delta u^+, \partial_n v^+ \right\rangle_e + \gamma h_e^{-1} \left\langle \partial_n u^+, \partial_n v^+ \right\rangle_e + \gamma h_e^{-3} \left\langle u^+, v^+ \right\rangle_e \Bigg) \\
- \sum_{e\in\mathcal{E}^I} \Bigg(& \left\langle \partial_n(\Delta v^+), u^- \right\rangle_e - \left\langle \Delta v^+, \partial_n u^- \right\rangle_e + \left\langle \partial_n(\Delta u^+), v^- \right\rangle_e - \left\langle \Delta u^+, \partial_n v^- \right\rangle_e \\
& + \gamma h_e^{-1} \left\langle \partial_n u^-, \partial_n v^+ \right\rangle_e + \gamma h_e^{-3} \left\langle u^-, v^+ \right\rangle_e + \gamma h_e^{-1} \left\langle \partial_n u^+, \partial_n v^- \right\rangle_e + \gamma h_e^{-3} \left\langle u^+, v^- \right\rangle_e \Bigg) \\
& + \sum_{e\in\mathcal{E}^I} \left(\gamma h_e^{-1} \left\langle \partial_n u^-, \partial_n v^- \right\rangle_e + \gamma h_e^{-3} \left\langle u^-, v^- \right\rangle_e \right)
\end{aligned} \tag{4.7}
$$

and for the Arnold formulation

$$
\begin{aligned}
a_h^\gamma(u,v) = \sum_{K\in\mathcal{T}_h}(\Delta u,\Delta v)_K + \sum_{e\in\mathcal{E}}\Bigg(&\frac{1}{2}\left\langle\partial_n(\Delta v^+),u^+\right\rangle_e - \frac{1}{2}\left\langle\Delta v^+,\partial_n u^+\right\rangle_e + \frac{1}{2}\left\langle\partial_n(\Delta u^+),v^+\right\rangle_e \\
&- \frac{1}{2}\left\langle\Delta u^+,\partial_n v^+\right\rangle_e + \gamma h_e^{-1}\left\langle\partial_n u^+,\partial_n v^+\right\rangle_e + \gamma h_e^{-3}\left\langle u^+,v^+\right\rangle_e\Bigg) \\
-\sum_{e\in\mathcal{E}^I}\Bigg(&\frac{1}{2}\left\langle\partial_n(\Delta v^+),u^-\right\rangle_e - \frac{1}{2}\left\langle\Delta v^+,\partial_n u^-\right\rangle_e + \frac{1}{2}\left\langle\partial_n(\Delta u^+),v^-\right\rangle_e - \frac{1}{2}\left\langle\Delta u^+,\partial_n v^-\right\rangle_e \\
&- \frac{1}{2}\left\langle\partial_n(\Delta v^-),u^+\right\rangle_e + \frac{1}{2}\left\langle\Delta v^-,\partial_n u^+\right\rangle_e - \frac{1}{2}\left\langle\partial_n(\Delta u^-),v^+\right\rangle_e + \frac{1}{2}\left\langle\Delta u^-,\partial_n v^+\right\rangle_e \\
&+ \gamma h_e^{-1}\left\langle\partial_n u^-,\partial_n v^+\right\rangle_e + \gamma h_e^{-3}\left\langle u^-,v^+\right\rangle_e + \gamma h_e^{-1}\left\langle\partial_n u^+,\partial_n v^-\right\rangle_e + \gamma h_e^{-3}\left\langle u^+,v^-\right\rangle_e\Bigg) \\
+\sum_{e\in\mathcal{E}^I}\Bigg(&\frac{1}{2}\left\langle\partial_n(\Delta v^-),u^-\right\rangle_e - \frac{1}{2}\left\langle\Delta v^-,\partial_n u^-\right\rangle_e + \frac{1}{2}\left\langle\partial_n(\Delta u^-),v^-\right\rangle_e - \frac{1}{2}\left\langle\Delta u^-,\partial_n v^-\right\rangle_e \\
&\gamma h_e^{-1}\left\langle\partial_n u^-,\partial_n v^-\right\rangle_e + \gamma h_e^{-3}\left\langle u^-,v^-\right\rangle_e\Bigg) \qquad (4.8)
\end{aligned}
$$

4.2 A Posteriori Error Estimation

4.2.1 Local A Posteriori Estimator

The setup for the local problem is exactly the same as in § 3.2.2, with appropriate changes to the bilinear form. Implementation and comparisons will be included in the final draft of this dissertation.

Again, first noting the integration by parts formula for a single element K

$$
\begin{aligned}
(\Delta^2 u, v)_K &= (\Delta(\Delta u), v)_K \\
&= -(\nabla(\Delta u), \nabla v)_K + \langle\partial_n\Delta u, v\rangle_{\partial K} \\
&= (\Delta u,\Delta v)_K - \langle\Delta u,\partial_n v\rangle_{\partial K} + \langle\partial_n\Delta u, v\rangle_{\partial K}
\end{aligned}
$$

implies

$$
(\Delta u,\Delta v)_K = (\Delta^2 u, v)_K + \langle\Delta u,\partial_n v\rangle_{\partial K} - \langle\partial_n\Delta u, v\rangle_{\partial K}\,.
$$

Therefore

$$
\begin{aligned}
(\Delta u,\Delta v)_K = (\Delta^2 u, v)_K &+ \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^B}}\left(\langle\Delta u,\partial_n v\rangle_e - \langle\partial_n\Delta u, v\rangle_e\right) \\
&+ \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I\\ K=K^+}}\left(\left\langle\Delta u^+,\partial_n v^+\right\rangle_e - \left\langle\partial_n\Delta u^+, v^+\right\rangle_e\right) - \sum_{\substack{e\in\partial K\\ e\in\mathcal{E}^I\\ K=K^-}}\left(\left\langle\Delta u^-,\partial_n v^-\right\rangle_e - \left\langle\partial_n\Delta u^-, v^-\right\rangle_e\right). \qquad (4.9)
\end{aligned}
$$

Now the local problem is

$$a'_K(\eta_K, v) = (f, v)_K - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^B}} \left(\left\langle g_D, \partial_n \Delta v + \gamma h_e^{-3} v \right\rangle_e + \left\langle g_N, \gamma h_e^{-1} \partial_n v - \Delta v \right\rangle_e \right) - a'(u_h^\gamma, v)$$

or

$$\begin{aligned} a'_K(\eta_K, v) = (f, v)_K &- \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^B}} \left(\left\langle g_D, \partial_n \Delta v + \gamma h_e^{-3} v \right\rangle_e + \left\langle g_N, \gamma h_e^{-1} \partial_n v - \Delta v \right\rangle_e \right) \\ &- \Big[(\Delta u_h^\gamma, \Delta v)_K + \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}}} \left(\left\langle [u_h^\gamma], \{\partial_n \Delta v\} \right\rangle_e - \left\langle [\partial_n u_h^\gamma], \{\Delta v\} \right\rangle_e \right. \\ &+ \left\langle \{\partial_n \Delta u_h^\gamma\}, [v] \right\rangle_e - \left\langle \{\Delta u_h^\gamma\}, [\partial_n v] \right\rangle_e + \gamma h_e^{-1} \left\langle [\partial_n u_h^\gamma], [\partial_n v] \right\rangle_e + \gamma h_e^{-3} \left\langle [u_h^\gamma], [v] \right\rangle_e \Big) \Big]. \end{aligned} \tag{4.10}$$

Substituting Eq 4.9 into Eq 4.10 gives

$$\begin{aligned} a'_K(\eta_K, v) = (f - \Delta^2 u_h^\gamma, v)_K &+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^B}} \left(\left\langle g_D - u_h^\gamma, \partial_n \Delta v + \gamma h_e^{-3} v \right\rangle_e + \left\langle g_N - \partial_n u_h^\gamma, \gamma h_e^{-1} \partial_n v - \Delta v \right\rangle_e \right) \\ - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^+}} \left(\left\langle \Delta (u_h^\gamma)^+, \partial_n v^+ \right\rangle_e - \left\langle \partial_n \Delta (u_h^\gamma)^+, v^+ \right\rangle_e \right) &+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^-}} \left(\left\langle \Delta (u_h^\gamma)^-, \partial_n v^- \right\rangle_e - \left\langle \partial_n \Delta (u_h^\gamma)^-, v^- \right\rangle_e \right) \\ - \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I}} \Bigg(&\left\langle [u_h^\gamma], \{\partial_n \Delta v\} \right\rangle_e - \left\langle [\partial_n u_h^\gamma], \{\Delta v\} \right\rangle_e + \left\langle \{\partial_n \Delta u_h^\gamma\}, [v] \right\rangle_e - \left\langle \{\Delta u_h^\gamma\}, [\partial_n v] \right\rangle_e \\ &+ \gamma h_e^{-1} \left\langle [\partial_n u_h^\gamma], [\partial_n v] \right\rangle_e + \gamma h_e^{-3} \left\langle [u_h^\gamma], [v] \right\rangle_e \Bigg). \end{aligned} \tag{4.11}$$

For the Baker formulation, Eq 4.11 becomes

$$\begin{aligned} a'_K(\eta_K, v) = (f - \Delta^2 u_h^\gamma, v)_K &+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^B}} \left(\left\langle g_D - u_h^\gamma, \partial_n \Delta v + \gamma h_e^{-3} v \right\rangle_e + \left\langle g_N - \partial_n u_h^\gamma, \gamma h_e^{-1} \partial_n v - \Delta v \right\rangle_e \right) \\ &+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^+}} \left(\left\langle [\partial_n u_h^\gamma], \Delta v^+ - \gamma h_e^{-1} \partial_n v^+ \right\rangle_e - \left\langle [u_h^\gamma], \partial_n \Delta v^+ + \gamma h_e^{-3} v^+ \right\rangle_e \right) \\ &+ \sum_{\substack{e \in \partial K \\ e \in \mathcal{E}^I \\ K = K^-}} \left(\left\langle \gamma h_e^{-1} [\partial_n u_h^\gamma] - [\Delta u_h^\gamma], \partial_n v^- \right\rangle_e + \left\langle [\partial_n \Delta u_h^\gamma] + \gamma h_e^{-3} [u_h^\gamma], v^- \right\rangle_e \right). \end{aligned} \tag{4.12}$$

For the Arnold formulation, Eq 4.11 becomes

$$
\begin{aligned}
a'_K(\eta_K, v) = (f - \Delta^2 u_h^\gamma, v)_K + \sum_{\substack{e\in\partial K \\ e\in\mathcal{E}^B}} \left(\left\langle g_D - u_h^\gamma, \partial_n \Delta v + \gamma h_e^{-3} v \right\rangle_e + \left\langle g_N - \partial_n u_h^\gamma, \gamma h_e^{-1} \partial_n v - \Delta v \right\rangle_e\right) \\
+ \sum_{\substack{e\in\partial K \\ e\in\mathcal{E}^I \\ K=K^+}} \left(\left\langle \frac{1}{2}\left[\partial_n \Delta u_h^\gamma\right] - \gamma h_e^{-3}[u_h^\gamma], v^+ \right\rangle_e - \left\langle \gamma h_e^{-1}[\partial_n u_h^\gamma] + \frac{1}{2}[\Delta u_h^\gamma], \partial_n v^+ \right\rangle_e \right. \\
\left. + \frac{1}{2}\left\langle [\partial_n u_h^\gamma], \Delta v^+ \right\rangle_e - \frac{1}{2}\left\langle [u_h^\gamma], \partial_n \Delta v^+ \right\rangle_e \right) \\
+ \sum_{\substack{e\in\partial K \\ e\in\mathcal{E}^I \\ K=K^-}} \left(\left\langle \frac{1}{2}\left[\partial_n \Delta u_h^\gamma\right] + \gamma h_e^{-3}[u_h^\gamma], v^- \right\rangle_e + \left\langle \gamma h_e^{-1}[\partial_n u_h^\gamma] - \frac{1}{2}[\Delta u_h^\gamma], \partial_n v^- \right\rangle_e \right. \\
\left. + \frac{1}{2}\left\langle [\partial_n u_h^\gamma], \Delta v^- \right\rangle_e - \frac{1}{2}\left\langle [u_h^\gamma], \partial_n \Delta v^- \right\rangle_e \right).
\end{aligned}
\tag{4.13}
$$

4.3 Estimator Performance

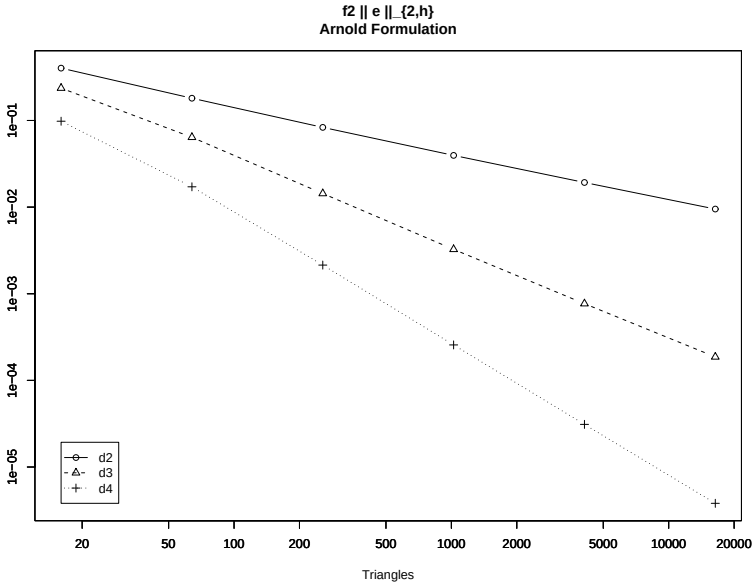
4.3.1 Effectivity Index Comparison: Arnold vs. Baker

Figures 4.1–4.3 show $\|e\|_{2,h}$ for functions f2, f3, and f4 under full refinement through 5 levels. Figures 4.4–4.6 show effectivity indices for the same set of runs. We utilize a heuristic residual-based type estimator, the effectivity index η_1 is calculated as

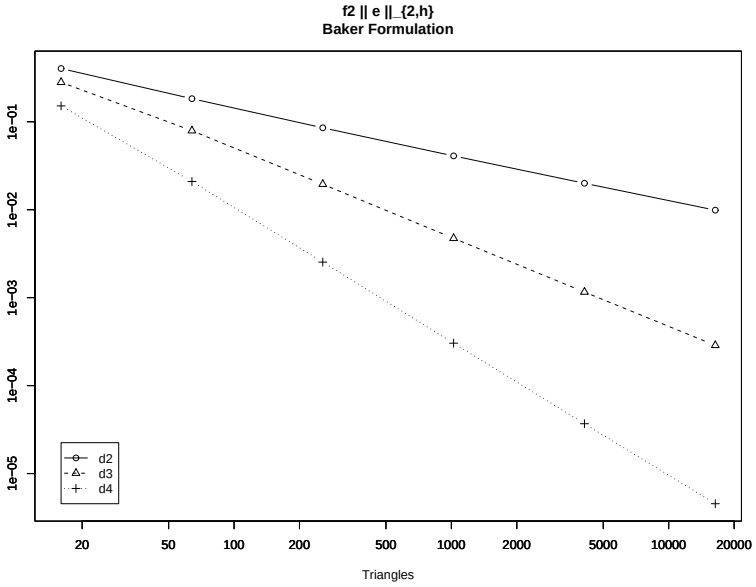
$$
\eta_1^2 = \frac{\xi^2}{\|e\|_{2,h}^2} \tag{4.14}
$$

where

$$
\begin{aligned}
\xi^2 = \sum_{K\in\mathcal{T}_h} h_K^4 \|f - \Delta^2 u_h^\gamma\|_K^2 + \sum_{e\in\mathcal{E}^I} \left(\gamma^2 h_e^{-1} |[\partial_n u_h^\gamma]|_e^2 + \right. \\
\left. \gamma^2 h_e^{-3} |[u_h^\gamma]|_e^2 + h_e |[\Delta u_h^\gamma]|_e^2 + h_e^3 |[\partial_n \Delta u_h^\gamma]|_e^2\right) \\
+ \sum_{e\in\mathcal{E}^B} \left(\gamma^2 h_e^{-1} |g_N - u_h^\gamma|_e^2 + \gamma^2 h_e^{-3} |g_D - u_h^\gamma|_e^2\right).
\end{aligned}
$$

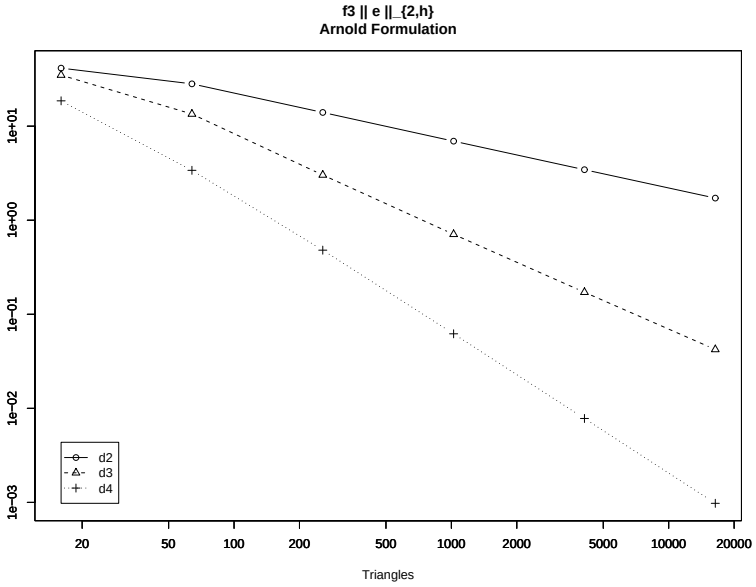


(a) f2 $||e||_{2,h}$ - Arnold

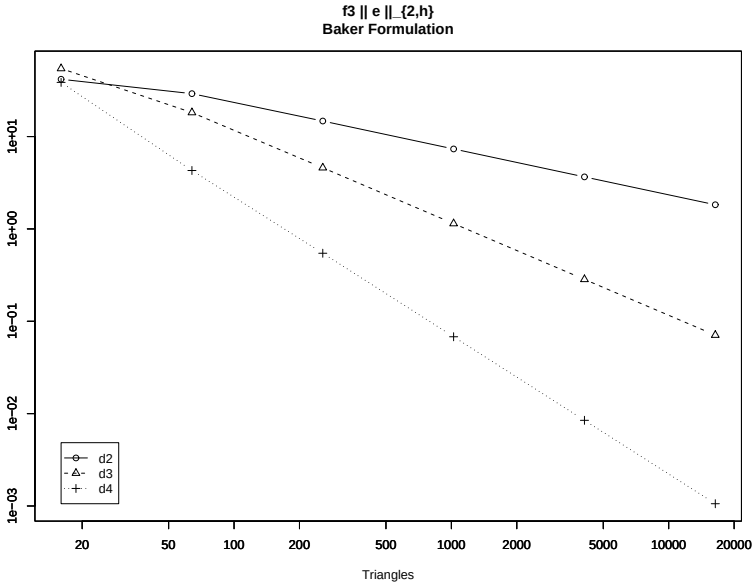


(b) f2 $||e||_{2,h}$ - Baker

Figure 4.1: f2 $||e||_{2,h}$

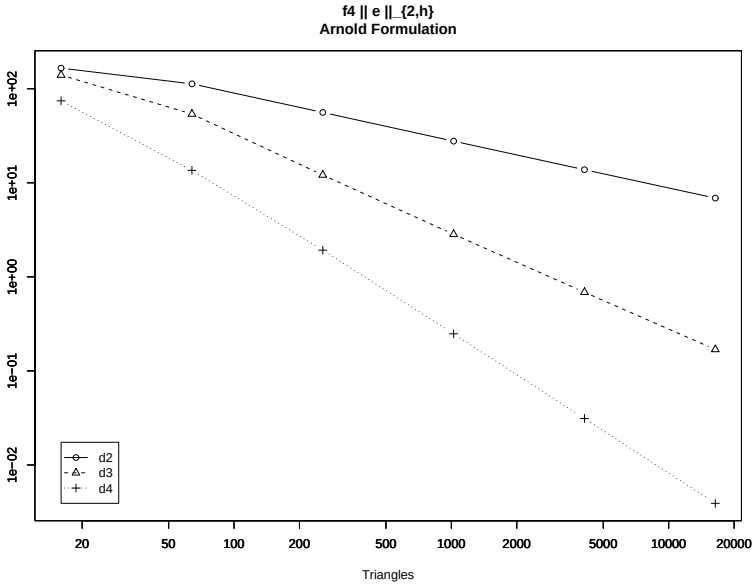


(a) f3 $||e||_{2,h}$ - Arnold

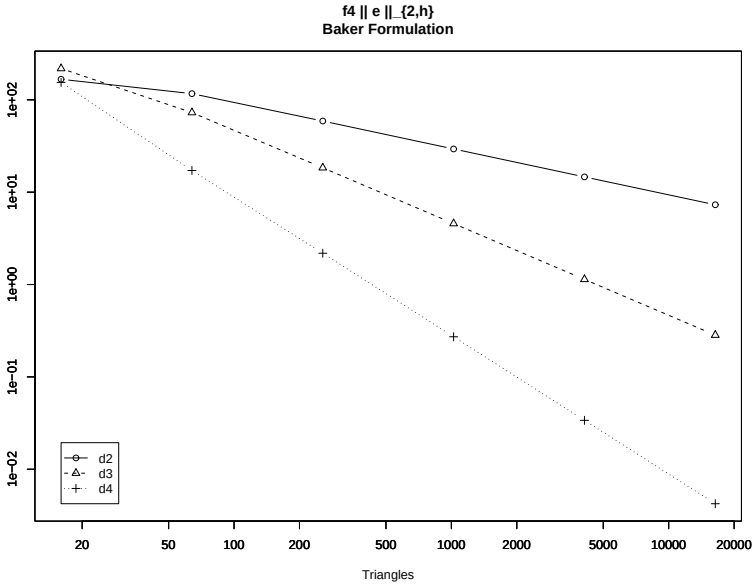


(b) f3 $||e||_{2,h}$ - Baker

Figure 4.2: f3 $||e||_{2,h}$

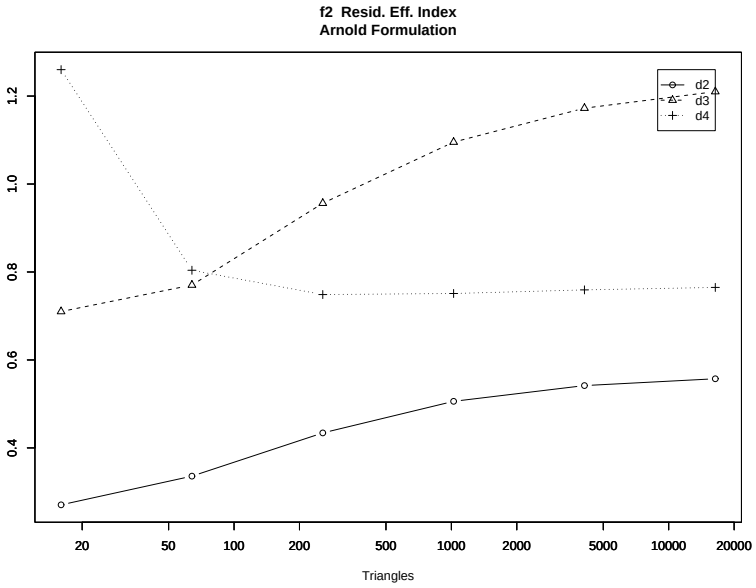


(a) f4 $||e||_{2,h}$ - Arnold

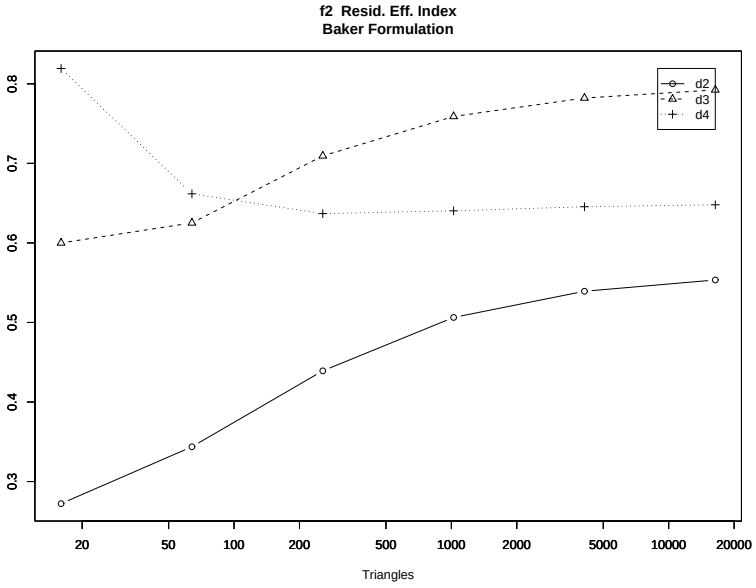


(b) f4 $||e||_{2,h}$ - Baker

Figure 4.3: f4 $||e||_{2,h}$

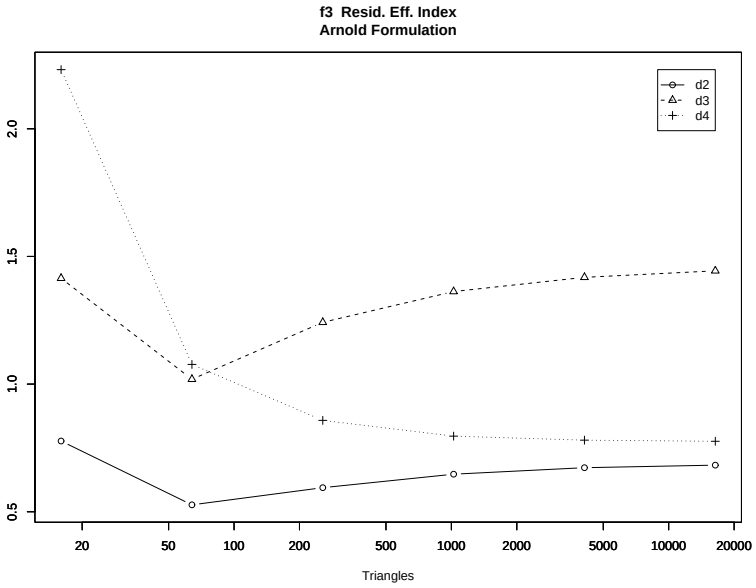


(a) Effectivity Index - f2, Arnold, Residual

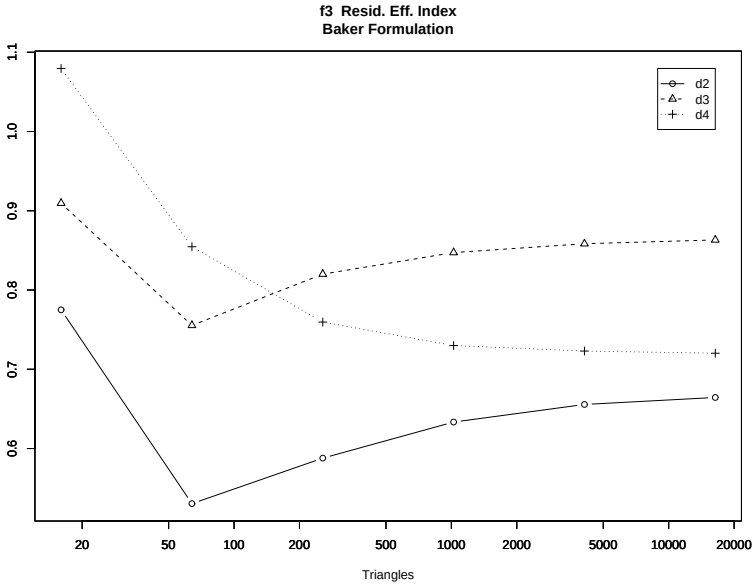


(b) Effectivity Index - f2, Baker, Residual

Figure 4.4: f2 Effectivity Indices

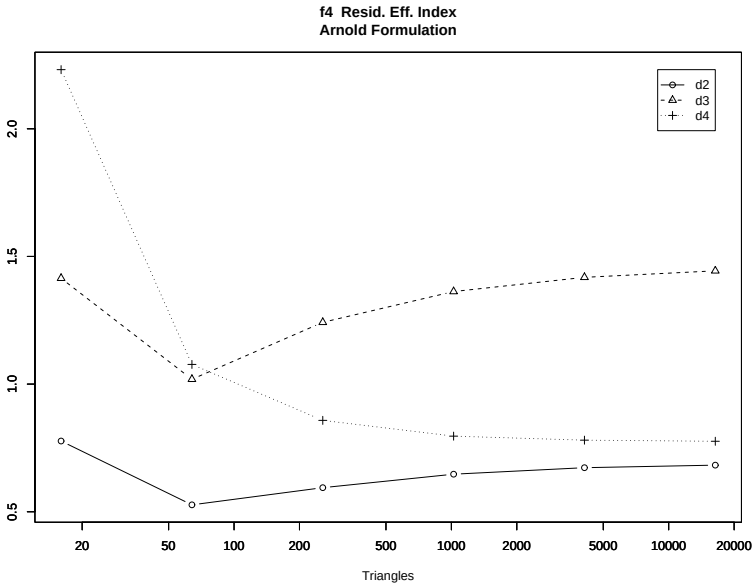


(a) Effectivity Index - f3, Arnold, Residual

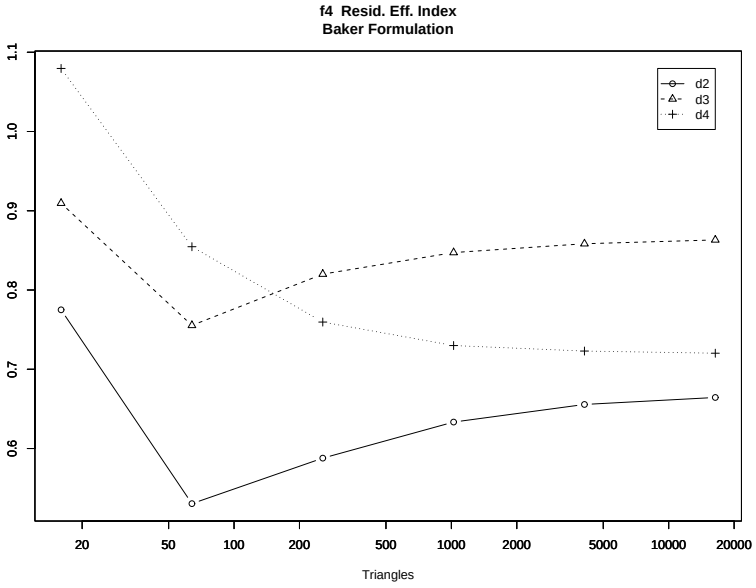


(b) Effectivity Index - f3, Baker, Residual

Figure 4.5: f3 Effectivity Indices



(a) Effectivity Index - f4, Arnold, Residual



(b) Effectivity Index - f4, Baker, Residual

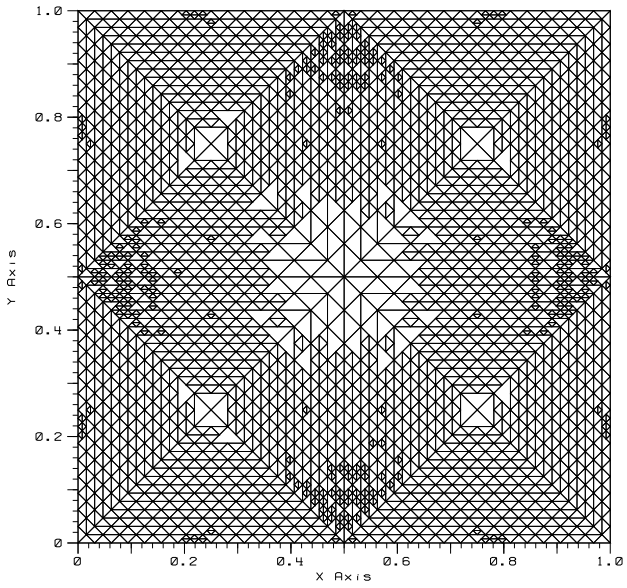
Figure 4.6: f4 Effectivity Indices

4.3.2 Adaptive Performance Comparison: Arnold vs. Baker

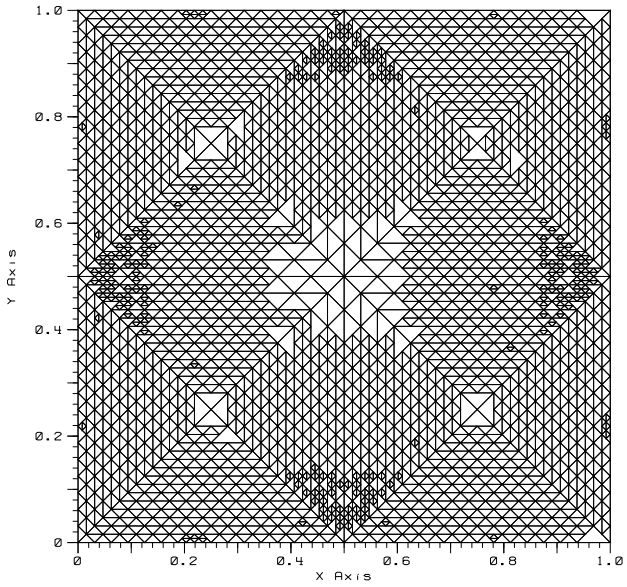
The following figures provide examples of adaptive meshes for some test problems. Table 4.1 illustrates the critical parameters used in generating these meshes.

Table 4.1: `Ell4` Adaptive Runs

run	γ	ϵ_{adapt}	dof	vertices	tri	aIter	Lvl	Figure
f2d2AR_A	6	0.00975	27078	2586	4513	12	5	4.7(a)
f2d2AR_B	6	0.01	26826	2505	4471	11	5	4.7(b)
f2d3AR_A	6	0.0017	26440	1699	2644	11	4	4.8(a)
f2d3AR_B	6	0.0017	26860	1685	2686	10	4	4.8(b)
f2d4AR_A	6	0.00012	25530	1041	1702	13	4	4.9(a)
f2d4AR_B	6	0.00012	26115	1040	1741	13	4	4.9(b)
f3d2AR_A	6	2.25	26484	2388	4414	13	5	4.10(a)
f3d2AR_B	6	2.35	26124	2365	4354	13	5	4.10(b)
f3d3AR_A	6	0.48	25090	1639	2509	10	4	4.11(a)
f3d3AR_B	6	0.48	25180	1639	2518	10	4	4.11(b)
f3d4AR_A	6	0.037	26295	981	1753	13	4	4.12(a)
f3d4AR_B	6	0.037	26070	957	1738	12	4	4.12(b)
f4d2AR_A	6	9.2	24918	2205	4153	13	5	4.13(a)
f4d2AR_B	6	9.2	27960	2575	4660	13	5	4.13(b)
f4d3AR_A	6	1.75	26740	1697	2674	9	4	4.14(a)
f4d3AR_B	6	1.75	26920	1705	2692	9	4	4.14(b)
f4d4AR_A	5	0.154	24405	920	1627	13	4	4.15(a)
f4d4AR_B	6	0.154	24405	904	1627	12	4	4.15(b)

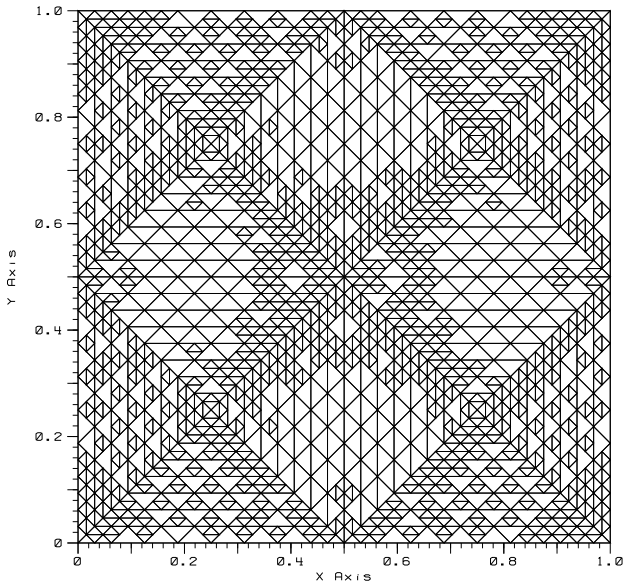


(a) Resid Est., Arnold

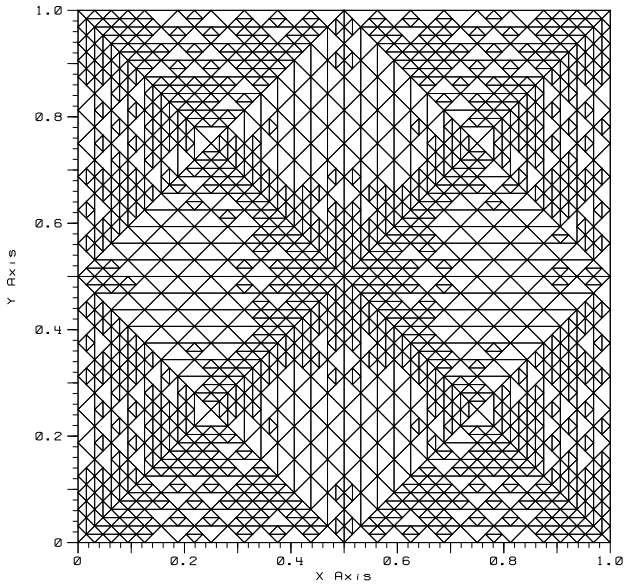


(b) Resid Est., Baker

Figure 4.7: Adaptive Meshes: f2, r=3

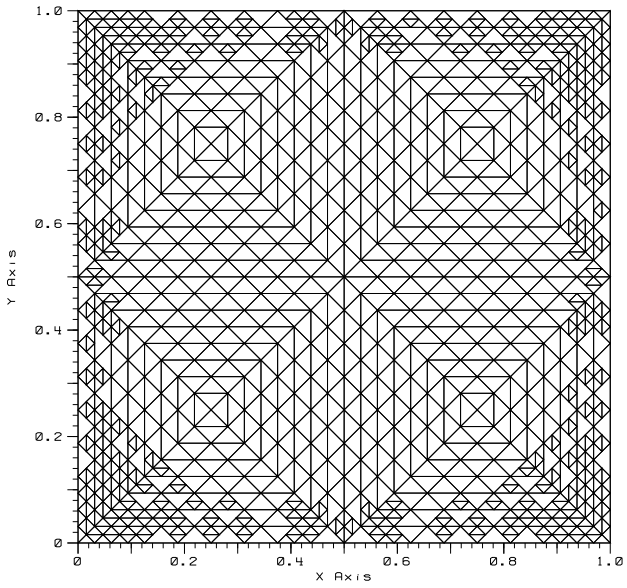


(a) Resid Est., Arnold

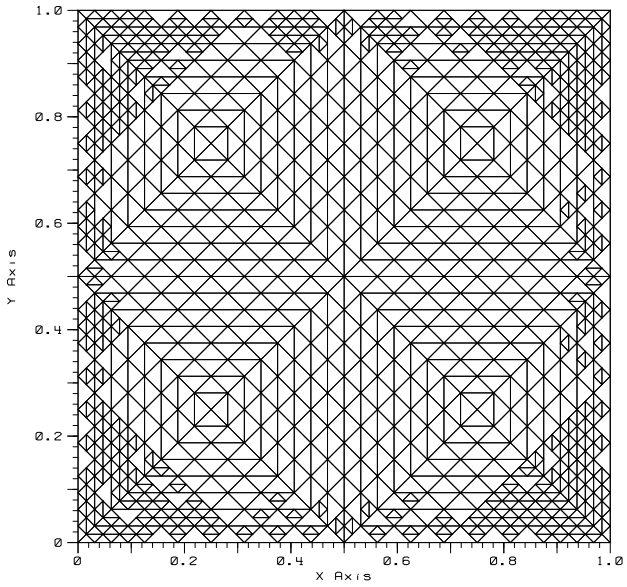


(b) Resid Est., Baker

Figure 4.8: Adaptive Meshes: f2, r=4

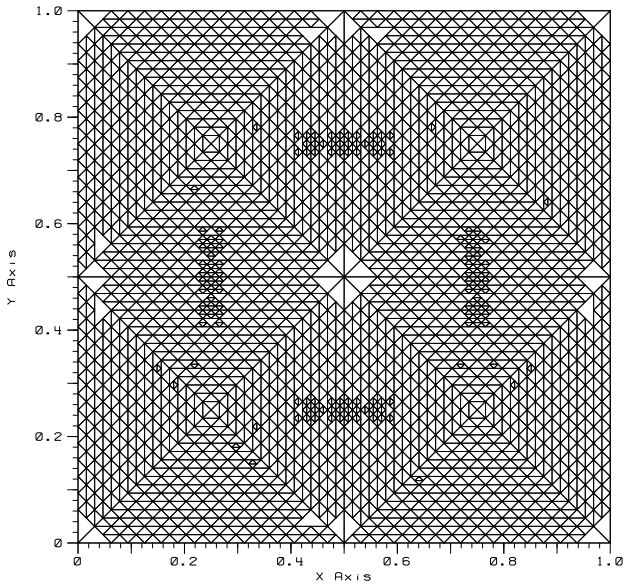


(a) Resid Est., Arnold

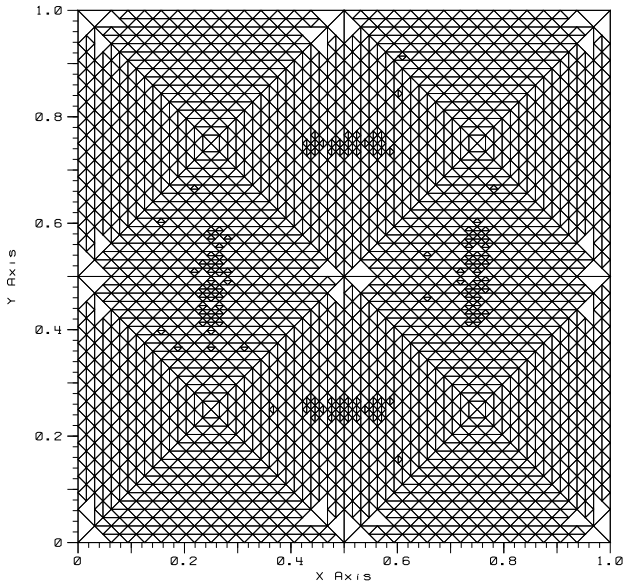


(b) Resid Est., Baker

Figure 4.9: Adaptive Meshes: f2, r=5

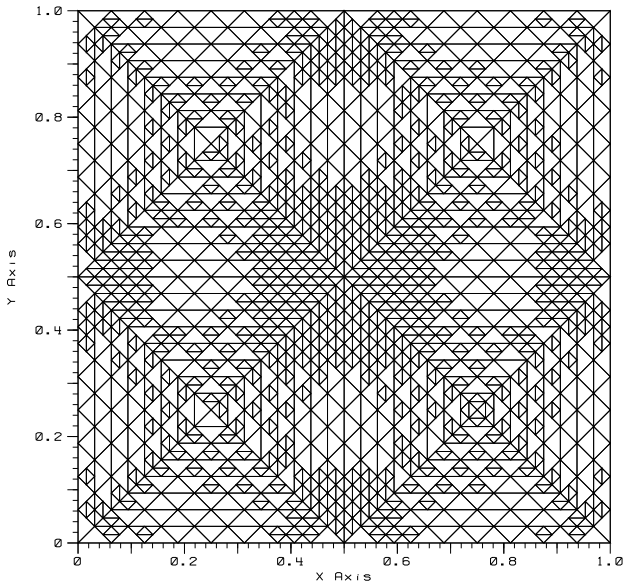


(a) Resid Est., Arnold

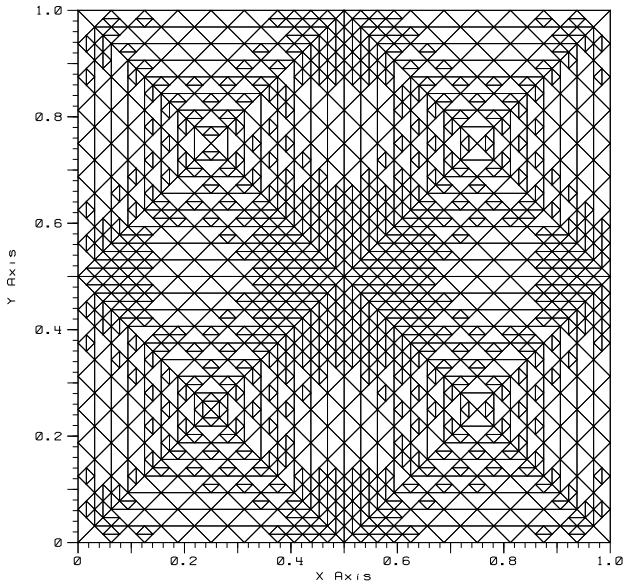


(b) Resid Est., Baker

Figure 4.10: Adaptive Meshes: f3, r=3

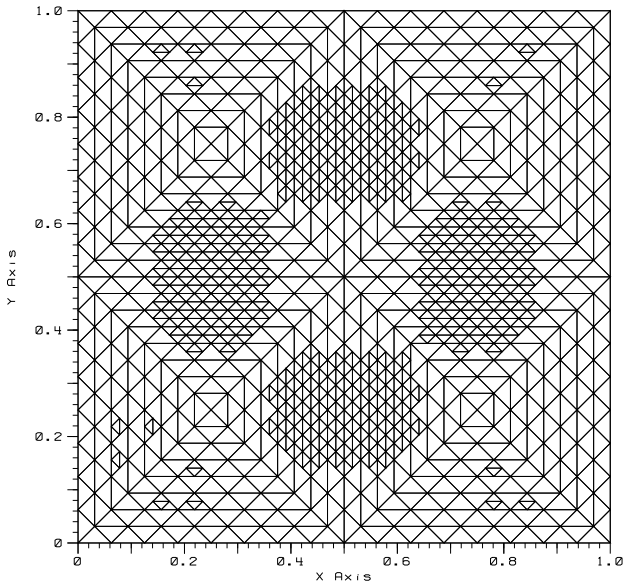


(a) Resid Est., Arnold

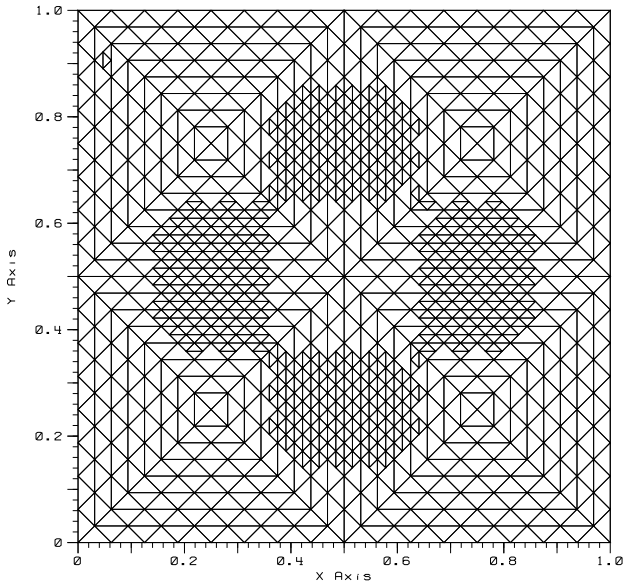


(b) Resid Est., Baker

Figure 4.11: Adaptive Meshes: f3, r=4

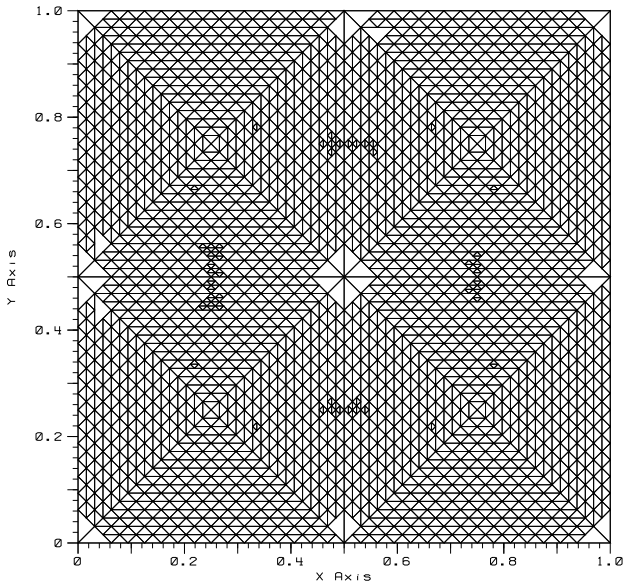


(a) Resid Est., Arnold

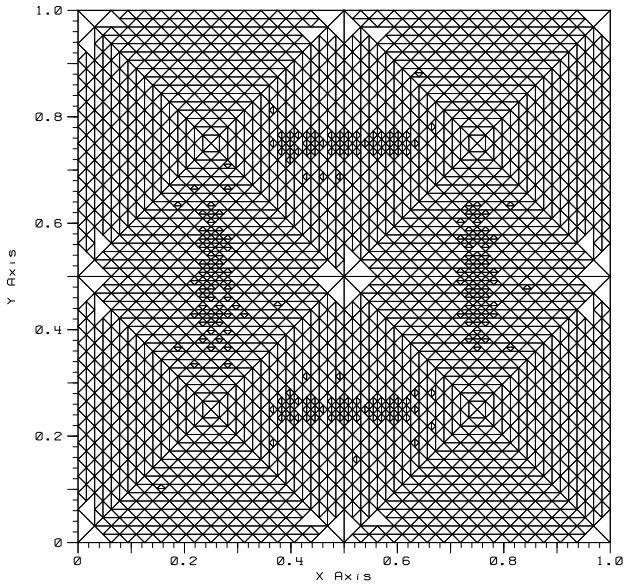


(b) Resid Est., Baker

Figure 4.12: Adaptive Meshes: f3, r=5

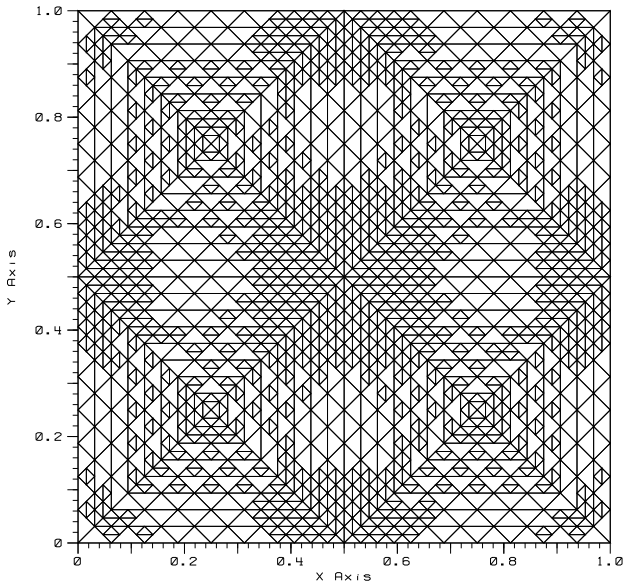


(a) Resid Est., Arnold

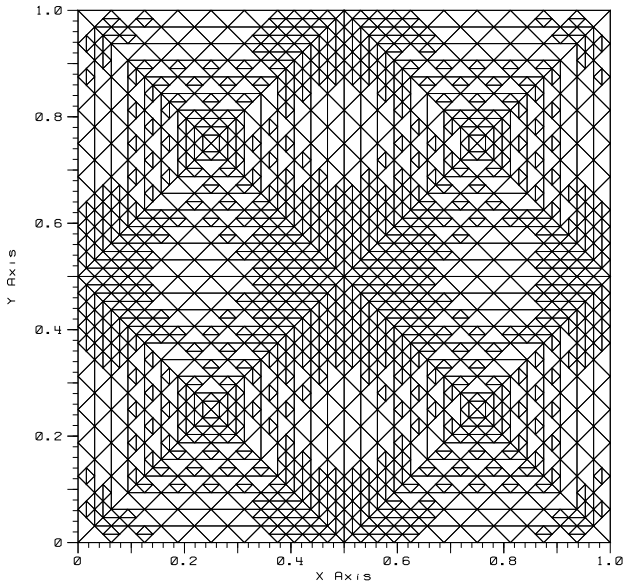


(b) Resid Est., Baker

Figure 4.13: Adaptive Meshes: f4, r=3

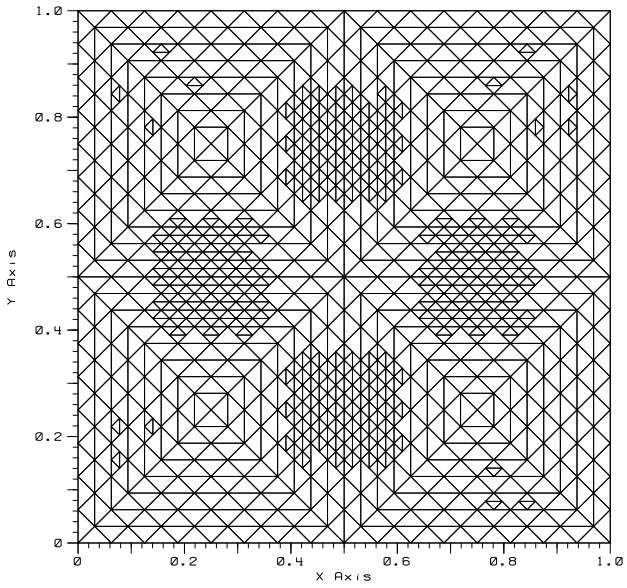


(a) Resid Est., Arnold

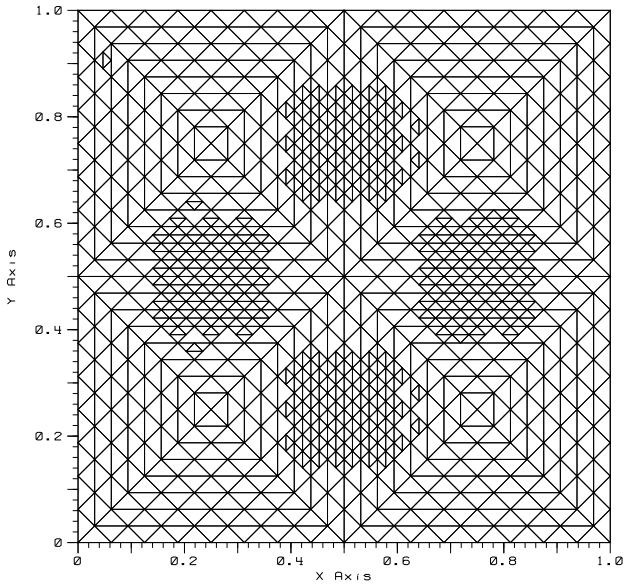


(b) Resid Est., Baker

Figure 4.14: Adaptive Meshes: f4, r=4



(a) Resid Est., Arnold



(b) Resid Est., Baker

Figure 4.15: Adaptive Meshes: f4, r=5

Chapter 5

Summary and Future Directions

This research has illustrated that the use of the DG method in an adaptive FEM environment has many attractive features from both a theoretical and computational point of view. The fact that there are no continuity constraints as those that exist in standard continuous Galerkin FEM approximations implies that one has great flexibility in how the individual elements are refined. This feature of DG has great potential for the future, from "drastic cutting" of an element more than one time per adaptive iteration to scalability and parallelization of the algorithms to solve really large problems on massively parallel clusters. While DG implementation requires one to solve for a larger number of unknowns than continuous Galerkin FEMs, it is felt that the performance improvements illustrated here and carried further, especially in the implementation and choice of solver will more than make up for the presence of these extra unknowns.

This research effort has often created more work to be done than could reasonably be achieved in the time frame to complete the dissertation, and thus there is plenty work to be tackled in the future. For example, in the adaptive mesh management there are still issues to be solved in adjusting inter hierarchy boundary levels based on how the adaptive problem's level storage requirements are changing. This would make the adaptive mesh management implementation even more powerful. In addition, extending the use of cache optimization to integrate more fully with the hierarchy mesh tree would also be of excellent value. The gains obtained from utilizing a modified block sparse row data format for operations on the related matrices and vectors should be pursued and more tightly integrated with the work being done with the Optimized Sparse Kernel Interface project (Vuduc et al., 2005).

We would like to extend the software developed here into a more flexible and useful tool for researchers. Included in this effort will be to extend the simple model problems to include advection terms as well as to be able to handle variable coefficient problems. An effort will be made to package the software developed here to make it "useable" for other researchers, i.e., automate selection of solver, optimization efforts, etc. Finally, we would also like to include nonlinear solvers in the software.

Determining an "optimal" choice for the penalty parameter γ under various PDE problem scenarios is definitely a goal for future work. In addition, developing sharp a posteriori estimates utilizing modifications of either the residual or local problem estimator is important to look at also, including local problem extension to increasing the local problem subspace to include both h and p refinement. Another area of future work is to work with the solution to the local problems as a "correction" to the solution; this aspect perhaps could be exploited and utilized in the solver implementation, specifically Multigrid.

We plan on pursuing extension of these methods to time dependent problems, such as the heat equations and the Cahn-Hilliard equations (Cahn and Hilliard, 1958; Feng and Karakashian, 2006). In addition, providing a more formal comparison between other current state of the art FEM implementations such as the excellent program `PLTMG` (Bank, 1998) and FEM toolbox `ALBERT` (Schmidt and Siebert, 2005) is also on the agenda for the future.

In conclusion, this research has provided me the best opportunity I could have asked for; taking what is the really beautiful mathematical theory of DG-FEM, applying fast and efficient state of the art algorithms, utilizing current existing state of the art software, all resulting in computer software that can be reused in the future to explore these methods even further.

Bibliography

Arnold, D. (1982). An interior penalty finite element method with discontinuous elements. *SIAM J. Num. Anal.*, 19:742–760.

Arnold, D., Brezzi, F., Cockburn, B., and Marini, L. (2002). Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Num. Anal.*, 39:1749–1779.

Babuşka, I. and Rheinboldt, W. (1978). Error estimates for adaptive finite element computations. *SIAM J. Numer. Anal.*, 15:736–754.

Babuşka, I. and Strouboulis, T. (2001). *Finite Element Method and its Reliability*. Numerical Mathematics and Computation. Oxford University Press, New York.

Baker, G. (1977). Finite element methods for elliptic equations using nonconforming elements. *Math. Comp.*, 31:45–59.

Baker, G., Jureidini, W., and Karakashian, O. (1990). Piecewise solenoidal vector fields and the stokes problem. *SIAM J. Num. Anal.*, 27:1466–1485.

Bank, R. E. (1998). *PLTMG: A software package for solving elliptic partial differential equations, Users' Guide 8.0*. SIAM, Philadelphia.

Bank, R. E. and Smith, K. (1993). A posteriori error estimates based on hierarchical bases. *SIAM J. Numer. Anal.*, 30:921–932.

Baumann, C. (1997). *An hp-adaptive Discontinuous Finite Element Method for Computational Fluid Dynamics*. PhD thesis, The University of Texas at Austin.

Baumann, C. and Oden, J. (1999). A discontinuous hp finite element method for convection–diffusion problems. Comput. Methods Appl. Mech. and Engng., In press, special issue on Spectral, Spectral Element and hp methods in CFD.

Beyls, K. (2004). *Software Methods to Improve Data Locality and Cache Behavior*. PhD thesis, Ghent University.

Bramble, J. H. (1993). *Multigrid Methods*, volume 294 of *Pitman Research Notes in Mathematical Sciences*. Longman Scientific & Technical, Essex, England.

Brenner, S. C. and Scott, L. R. (2004). *The Mathematical Theory of Finite Element Methods*. Springer Verlag, New York, second edition.

Browne, S., Dongarra, J., Garner, N., Ho, G., and Mucci, P. (2000). A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204.

Cahn, J. and Hilliard, J. (1958). Free energy of a nonuniform system i.interfacial free energy. *J. Chem. Phys.*, 28:258–267.

Ciarlet, P. G. (1978). *The Finite Element Method for Elliptic Problems*. North-Holland, Amsterdam,New York,Oxford.

Cockburn, B. (1997). An introduction to the discontinuous Galerkin method for convection dominated problems. In Quarteroni, A., editor, *Advanced numerical approximation of nonlinear hyperbolic equations*, volume 1697 of *Lecture Notes in Mathematics; subseries Fondazione C.I.M.E., Firenze*, pages 151–268. Springer Verlag.

Cockburn, B., Karniadakis, G., and Shu, C. (2000). *Discontinuous Galerkin methods, Theory, Computation and Applications*. Springer lecture notes in computational science and engineering. Springer Verlag.

Cockburn, B. and Shu, C. (1998). The local discontinuous Galerkin finite element method for convection-diffusion systems. *SIAM JNA*, 35:2440–2463.

Cockburn, B. and Shu, editors, C. (2005). Special issue on discontinuous Galerkin methods. *Jour. of Scient. Comput.*, 22–23.

Dörfler, W. (1996). A convergent adaptive algorithm for poisson's equation. *SIAM J. Numer. Anal.*, 33:1106–1124.

Douglas, C. C., Hu, J., Kowarschik, M., Rüde, U., and Weiss, C. (2000). Cache optimization for structured and unstructured grid multigrid. *Elect. Trans. Numer. Anal.*, 10:21–40.

Douglas Jr., J. and Dupont, T. (1976). Interior penalty procedures for elliptic and parabolic Galerkin methods. In *Lecture Notes in Physics*, volume 58. Springer Verlag, Berlin.

Eriksson, K. and Johnson, C. (1998). An adaptive finite element method for linear elliptic problems. *Math. Comp.*, 50:361–382.

Feng, X. and Karakashian, O. (2001). Two-level additive schwarz methods for a discontinuous Galerkin approximation of second-order elliptic problems. *SIAM J. Numer. Anal.*, 39:1343–1365.

Feng, X. and Karakashian, O. (2006). Fully discrete dynamic mesh discontiuous Galerkin methods for the cahn-hilliard equation of phase transition. *Math. Comp. Submitted*.

Goedecker, S. and Hoisie, A. (2001). *Performance Optimization of Numerically Intensive Codes*. SIAM, Philadelphia.

Golub, G. H. and van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, third edition.

Greenbaum, A. (1997). *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia.

Greenstadt, J. (1982). The cell discretization algorithm for elliptic partial differential equations. *SIAM J. Sci. Stat. Comput.*, 3:261–288.

Hackbusch, W. (1985). *Multigrid Methods and Applications*, volume 4 of *Computational Mathematics*. Springer–Verlag, Berlin.

Johnson, C. (1992). *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, Cambridge.

Karakashian, O. and Jureidini, W. (1998). A nonconforming finite element method for the stationary navier-stokes equations. *SIAM J. Num. Anal.*, 35:93–120.

Karakashian, O. and Pascal, F. (2003). A posteriori error estimates for a discontinuous Galerkin approximation of second-order elliptic equations. *SIAM J. Num. Anal.*, 41:2374–2399.

Karakashian, O. and Pascal, F. (2004). Adaptive discontinuous Galerkin approximations of second-order elliptic equations. In et al., P. N., editor, *In Proceedings of the European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS 2004*, Jyväskylä, Finland.

Karakashian, O. and Pascal, F. (2006). Adaptive discontinuous Galerkin approximations of second-order elliptic problems. *SIAM J. Numer. Anal. (to appear)*.

Karypis, G. and Kumar, V. (1995). *MeTis Users Manual: Unstrctured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*.

Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392.

Kernighan, B. and Ritchie, D. (1978). *The* `C` *Programming Language*. P T R Prentice Hall, Englewood Cliffs, New Jersey.

Kowarschik, M. (2004). *Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchial Memory Architechtures*. PhD thesis, Der Technischen Fakultät der Universität Erlangen-Nürnberg.

Morin, P., Nochetto, R., and Siebert, K. (2000). Data oscillation and convergence of adaptive fem. *SIAM J. Numer. Anal.*, 38:466–488.

Nitsche, J. (1971). Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind. *Abh. Math. Sem.*, 36:9–15.

Prudhomme, S., Pascal, F., Oden, J., and Romkes, A. (2000). Review of A Priori error estimation for discontinuous Galerkin methods. Technical Report TICAM Report 00-27, Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin.

Reed, W. and Hill, T. (1973). Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory.

Rivière, B., Wheeler, M., and Girault, V. (1999). Improved energy estimates for interior penalty constrained and discontinuous Galerkin methods for elliptic problems I. *Comput. Geosci.*, 3:337–360.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems, 2nd edition*. SIAM, Philadelpha, PA.

Schmidt, A. and Siebert, K. G. (2005). *Design of Adaptive Finite Element Software:The Finite Element Toolbox ALBERTA*, volume 42 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Lin, M. C. and Manocha, D., editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag. From the First ACM Workshop on Applied Computational Geometry.

Stevens, W. R. (1993). *Advanced Programming in the UNIX Environment*. Addison-Wesley, Boston.

Strout, M. M., Carter, L., and Ferrante, J. (2001). Rescheduling for locality in sparse matrix computations. In *Computational Science - ICCS 2001, International Conference, San Francisco, CA, USA, May 28-30, 2001. Proceedings, Part I*, volume 2073 of *Lecture Notes in Computer Science*, pages 137–148. Springer.

Varga, R. S. (1962). *Matrix Iterative Analysis*. Prentice–Hall, Englewood Cliffs, NJ.

Verfürth, R. (1995). *A review of A posteriori error Estimation and Adaptive Mesh Refinement Techniques*. Wiley-Teubner, New York.

Verfürth, R. (1998). A posteriori error estimators for convection-diffusion equations. *Numer. Math.*, 80:641–663.

Vo, K.-P. (1996). Vmalloc: A general and efficient memory allocator. *Software Practice & Experience*, 26:1–18.

Vuduc, R., Demmel, J. W., and Yelick, K. A. (2005). OSKI: A library of automatically tuned sparse matrix kernels. In *Proceedings of SciDAC 2005*, Journal of Physics: Conference Series, San Francisco, CA, USA. Institute of Physics Publishing. (*to appear*).

Weiss, C. (2001). *Data Locality Optimizations for Multigrid Methods on Structured Grids*. PhD thesis, Institut für Informatik, Technische Universität München, Munich, Germany.

Whaley, R. C., Petitet, A., and Dongarra, J. J. (2001). Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).

Wheeler, M. (1978). An elliptic collocation-finite element method with interior penalties. *SIAM J. Num. Anal.*, 15:152–161.

Young, D. M. (1971). *Iterative Solution of Large Linear Systems*. Academic Press, New York.

Appendices

Appendix A

`Ell2` Stiffness Matrix Assembly Detail

Diagonal Blocks: Dirichlet term.

Assume that on K the following functions are defined

$$u(\mathbf{x}) = \sum_{j=1}^{N} u(\mathbf{x}_j^K)\phi_j^K(\mathbf{x})$$

and

$$v(\mathbf{x}) = \sum_{i=1}^{N} \phi_i^K(\mathbf{x})$$

where N is the number of degrees of freedom and for convenience we will drop the superscript K annotation unless otherwise needed. Then

$$\begin{aligned}
(\nabla u(\mathbf{x}), \nabla v(\mathbf{x}))_K &= \int_K \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x})\, d\mathbf{x} \\
&= \sum_{i,j=1}^{N} u(\mathbf{x}_j^K) \int_K \nabla \phi_j \cdot \nabla \phi_i \, d\mathbf{x} \\
&= \sum_{i,j=1}^{N} u(\mathbf{x}_j^K) \int_K \left(\frac{\partial \phi_j}{\partial x}\frac{\partial \phi_i}{\partial x} + \frac{\partial \phi_j}{\partial y}\frac{\partial \phi_i}{\partial y} \right) d\mathbf{x} \\
&= \sum_{i,j=1}^{N} (\nabla \phi_j, \nabla \phi_i)_K \, u(\mathbf{x}_j^K) = \sum_{i,j=1}^{N} (\nabla \phi_j, \nabla \phi_i)_K \, \alpha_j^K .
\end{aligned}$$

where $\{\alpha_j^K\}_{j=1}^N$ are the unknowns (dofs) for triangle K. Applying change of variable transformations for quadrature over the reference element $\hat{K}$ implies

$$\frac{\partial\phi_j}{\partial x}\frac{\partial\phi_i}{\partial x}+\frac{\partial\phi_j}{\partial y}\frac{\partial\phi_i}{\partial y}=\frac{1}{(2|K|)^2}\left[\left(\frac{\partial\hat{\phi}_j}{\partial\hat{x}}\hat{a}_{11}+\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\hat{a}_{21}\right)\left(\frac{\partial\hat{\phi}_i}{\partial\hat{x}}\hat{a}_{11}+\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\hat{a}_{21}\right)+\left(\frac{\partial\hat{\phi}_j}{\partial\hat{x}}\hat{a}_{12}+\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\hat{a}_{22}\right)\left(\frac{\partial\hat{\phi}_i}{\partial\hat{x}}\hat{a}_{12}+\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\hat{a}_{22}\right)\right].$$

Thus, with N_q being the number of quadrature points for a chosen quadrature rule and evaluating each partial derivative at quadrature points q, $q=1,\ldots,N_q$ with quadrature weight $\hat{w}_q$,

$$(\nabla\phi_j,\nabla\phi_i)_K=\left(\frac{1}{2|K|}\right)^2\left\{(2|K|)\sum_{q=1}^{N_q}\hat{w}_q\left[\left(\frac{\partial\hat{\phi}_j}{\partial\hat{x}}\hat{a}_{11}+\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\hat{a}_{21}\right)\left(\frac{\partial\hat{\phi}_i}{\partial\hat{x}}\hat{a}_{11}+\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\hat{a}_{21}\right)+\left(\frac{\partial\hat{\phi}_j}{\partial\hat{x}}\hat{a}_{12}+\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\hat{a}_{22}\right)\left(\frac{\partial\hat{\phi}_i}{\partial\hat{x}}\hat{a}_{12}+\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\hat{a}_{22}\right)\right]_q\right\}$$

or

$$(\nabla\phi_j,\nabla\phi_i)_K=\frac{1}{2|K|}\left\{\sum_{q=1}^{N_q}\hat{w}_q\left[\left(\frac{\partial\hat{\phi}_j}{\partial\hat{x}}\hat{a}_{11}+\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\hat{a}_{21}\right)\left(\frac{\partial\hat{\phi}_i}{\partial\hat{x}}\hat{a}_{11}+\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\hat{a}_{21}\right)+\left(\frac{\partial\hat{\phi}_j}{\partial\hat{x}}\hat{a}_{12}+\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\hat{a}_{22}\right)\left(\frac{\partial\hat{\phi}_i}{\partial\hat{x}}\hat{a}_{12}+\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\hat{a}_{22}\right)\right]_q\right\}.$$

Note that

$$A_{K,1}:=\Big((\nabla\phi_j,\nabla\phi_i)_K\Big)_{i,j=1}^N \tag{A.1}$$

is a symmetric matrix and is the *Dirichlet* portion of the diagonal block matrix A_K.

Diagonal Blocks: Flux terms.

Associated with each interior edge $e\in\mathcal{E}^I$ are two triangles, (K^+,K^-). The edge integrals which contribute to the diagonal blocks relating to fluxes are $\langle\partial_n u^+,v^+\rangle_e$, $\langle\partial_n v^+,u^+\rangle_e$ for K^+ and $\langle\partial_n u^-,v^-\rangle_e$, $\langle\partial_n v^-,u^-\rangle_e$ for K^- where n is the unit outward normal to K^+.

First consider the term $\langle \partial_n v^+, u^+ \rangle_e$:

$$\begin{aligned}\langle \partial_n v^+, u^+ \rangle_e &= \left\langle \sum_{j=1}^{N} \alpha_j^+ \phi_j^+, \nabla v^+ \cdot n \right\rangle_e \\ &= \sum_{j=1}^{N} \alpha_j^+ \left\langle \phi_j^+, \sum_{i=1}^{N} \partial_n \phi_i^+ \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \phi_j^+, \partial_n \phi_i^+ \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\left\langle \phi_j^+, \partial_n \phi_i^+ \right\rangle_e = \frac{h_e}{2|K|} \sum_{q=1}^{N_q} \hat{w}_q \left\{ \hat{\phi}_j \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right] \right\}_q .$$

Now consider the term $\langle \partial_n u^+, v^+ \rangle_e$:

$$\begin{aligned}\langle \partial_n u^+, v^+ \rangle_e &= \left\langle \sum_{j=1}^{N} \phi_i^+, \nabla u^+ \cdot n \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \phi_i^+, \sum_{j=1}^{N} \alpha_j^+ \partial_n \phi_j^+ \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \phi_i^+, \partial_n \phi_j^+ \right\rangle_e .\end{aligned}$$

Applying the appropriate edge quadrature rule implies that

$$\left\langle \phi_i^+, \partial_n \phi_j^+ \right\rangle_e = \frac{h_e}{2|K|} \sum_{q=1}^{N_q} \hat{w}_q \left\{ \hat{\phi}_i \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right] \right\}_q .$$

Note that similar formulae exist for the terms $\langle \partial_n u^-, v^- \rangle_e$, $\langle \partial_n v^-, u^- \rangle_e$.

For the Baker formulation (Eq. 3.13)

$$A_{K,2} := \left(\sum_{\substack{e \in \partial K \\ K = K^+ \\ e \in \mathcal{E}^I \cup \mathcal{E}_D^B}} \left(\left\langle \phi_j^+, \partial_n \phi_i^+ \right\rangle_e + \left\langle \phi_i^+, \partial_n \phi_j^+ \right\rangle_e \right) \right)_{i,j=1}^{N} \tag{A.2}$$

is a symmetric matrix and contributes to the *Flux* portion of the diagonal block matrix A_K. For the

Arnold formulation (Eq. 3.14)

$$A^+_{K,2} := \left(\sum_{\substack{e\in\partial K \\ K=K^+ \\ e\in\mathcal{E}^I}} \frac{1}{2}\left(\langle \phi^+_j, \partial_n \phi^+_i \rangle_e + \langle \phi^+_i, \partial_n \phi^+_j \rangle_e \right) \right)_{i,j=1}^N + \left(\sum_{\substack{e\in\partial K \\ K=K^+ \\ e\in\mathcal{E}^B_D}} \left(\langle \phi^+_j, \partial_n \phi^+_i \rangle_e + \langle \phi^+_i, \partial_n \phi^+_j \rangle_e \right) \right)_{i,j=1}^N \tag{A.3}$$

and

$$A^-_{K,2} := \left(\sum_{\substack{e\in\partial K \\ K=K^- \\ e\in\mathcal{E}^I}} \frac{1}{2}\left(\langle \phi^-_j, \partial_n \phi^-_i \rangle_e + \langle \phi^-_i, \partial_n \phi^-_j \rangle_e \right) \right)_{i,j=1}^N \tag{A.4}$$

both contribute to A_K.

Diagonal Blocks: Penalty terms.

Penalty terms $\gamma h_e^{-1}\langle u^+, v^+\rangle_e$ and $\gamma h_e^{-1}\langle u^-, v^-\rangle_e$ also contribute to the formulation of the diagonal block A_K. Following a similar path to what was done in the previous section

$$\gamma h_e^{-1}\langle u^+, v^+\rangle_e = \gamma h_e^{-1} \sum_{i,j=1}^N \alpha^+_j \langle \phi^+_j, \phi^+_i\rangle_e$$

with

$$\langle \phi^+_j, \phi^+_i\rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q [\hat{\phi}_j \hat{\phi}_i]_q$$

implies

$$\gamma h_e^{-1}\langle \phi^+_j, \phi^+_i\rangle_e = \gamma \sum_{q=1}^{N_q} \hat{w}_q [\hat{\phi}_j \hat{\phi}_i]_q.$$

Note here that the h_e^{-1} weighting factor is absorbed into the edge quadrature. Similarly for K^-

$$\gamma h_e^{-1}\langle \phi^-_j, \phi^-_i\rangle_e = \gamma \sum_{q=1}^{N_q} \hat{w}_q [\hat{\phi}_j \hat{\phi}_i]_q.$$

Therefore, the penalty contributions to A_K can be represented as

$$A^+_{K,3} := \Bigg(\sum_{\substack{e\in\partial K \\ K=K^+ \\ e\in\mathcal{E}^I\cup\mathcal{E}^B_D}} \gamma h_e^{-1} \langle \phi_j^+, \phi_i^+ \rangle_e \Bigg)_{i,j=1}^N \tag{A.5}$$

and

$$A^-_{K,3} := \Bigg(\sum_{\substack{e\in\partial K \\ K=K^- \\ e\in\mathcal{E}^I}} \gamma h_e^{-1} \langle \phi_j^-, \phi_i^- \rangle_e \Bigg)_{i,j=1}^N \tag{A.6}$$

Diagonal Block Assembly.

For the Baker formulation, K^- contributions to A_K only occur through the penalty terms $A^-_{K,3}$. For the Arnold formulation, contributions to A_K involve the flux terms $A^-_{K,2}$ and the penalty terms $A^-_{K,3}$. Since DG allows the presence of hanging nodes, special care must be taken with respect to accumulation of A_K components due to K^- contributions. In the case where a hanging node is present along an edge $e \in \partial K$ and $K = K^-$, then $A^-_{K,3}$ terms will accumulate in A_K from all (up to two) triangles adjacent to K along e

For each interior edge $e \in \partial K$ define a *hanging node edge weight* σ_e^K

$$\sigma_e^K := \begin{cases} 1 & Lvl(K^+) \le Lvl(K^-) \\ 2 & Lvl(K^+) > Lvl(K^-). \end{cases}$$

Therefore, $A^-_{K,3}$ can be rewritten to take hanging nodes into account as

$$A^-_{K,3} := \Bigg(\sum_{\substack{e\in\partial K \\ K=K^- \\ e\in\mathcal{E}^I}} \sigma_e^K \gamma h_e^{-1} \langle \phi_j^-, \phi_i^- \rangle_e \Bigg)_{i,j=1}^N \tag{A.7}$$

Note that we assume here that if $K = K^-$ and along edge $e \in \partial K$ there are two neighboring triangles (i.e., edge e is split into two edges e_1, e_2), then both neighboring triangles K_{e_1}, K_{e_2} are *both* considered to be $K^+_{e_1}, K^+_{e_2}$ relative to e_1, e_2 respectively.

Then for all $K \in \mathcal{T}_h$, assembly of A_K can be described for the Baker formulation as

$$A_K = A_{K,1} - A_{K,2} + A^+_{K,3} + A^-_{K,3} \tag{A.8}$$

and for the Arnold formulation as

$$A_K = A_{K,1} - A^+_{K,2} + A^-_{K,2} + A^+_{K,3} + A^-_{K,3}. \tag{A.9}$$

Off-Diagonal Blocks: Flux Mix Terms.

The edge integrals which contribute to the off-diagonal blocks relating to fluxes are $\langle \partial_n u^+, v^- \rangle_e$, $\langle \partial_n v^+, u^- \rangle_e$, $\langle \partial_n u^-, v^+ \rangle_e$, and $\langle \partial_n v^-, u^+ \rangle_e$, which describe the *mixing* of fluxes through edge e.

Now consider the term $\langle \partial_n u^+, v^- \rangle_e$:

$$\begin{aligned}
\langle \partial_n u^+, v^- \rangle_e &= \left\langle \nabla u^+ \cdot n, \sum_{i=1}^{N} \phi_i^- \right\rangle_e \\
&= \sum_{i=1}^{N} \left\langle \sum_{j=1}^{N} \alpha_j^+ \partial_n \phi_j^+, \phi_i^- \right\rangle_e \\
&= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \partial_n \phi_j^+, \phi_i^- \right\rangle_e .
\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}
&\left\langle \partial_n \phi_j^+, \phi_i^- \right\rangle_e \\
&\quad = \frac{h_e}{2|K^+|} \sum_{q=1}^{N_q} \hat{w}_q \left\{ \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right]_q \hat{\phi}_i \Big|_{N_q - q + 1} \right\}.
\end{aligned}$$

Now for the term $\langle \partial_n u^-, v^+ \rangle_e$

$$\begin{aligned}
\langle \partial_n u^-, v^+ \rangle_e &= \left\langle \nabla u^- \cdot n, \sum_{i=1}^{N} \phi_i^+ \right\rangle_e \\
&= \sum_{i=1}^{N} \left\langle \sum_{j=1}^{N} \alpha_j^- \partial_n \phi_j^-, \phi_i^+ \right\rangle_e \\
&= \sum_{i,j=1}^{N} \alpha_j^- \left\langle \partial_n \phi_j^-, \phi_i^+ \right\rangle_e .
\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}
&\left\langle \partial_n \phi_j^-, \phi_i^+ \right\rangle_e \\
&\quad = \frac{h_e}{2|K^-|} \sum_{q=1}^{N_q} \hat{w}_q \left\{ \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right]_{N_q - q + 1} \hat{\phi}_i \Big|_q \right\}.
\end{aligned}$$

Now consider the term $\langle \partial_n v^+, u^- \rangle_e$:

$$\begin{aligned}\langle \partial_n v^+, u^- \rangle_e &= \left\langle \nabla v^+ \cdot n, \sum_{j=1}^{N} \alpha_j^- \phi_j^- \right\rangle_e \\ &= \sum_{j=1}^{N} \alpha_j^- \left\langle \sum_{i=1}^{N} \partial_n \phi_i^+, \phi_j^- \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^- \left\langle \partial_n \phi_i^+, \phi_j^- \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}&\left\langle \partial_n \phi_i^+, \phi_j^- \right\rangle_e \\ &\quad = \frac{h_e}{2|K^+|} \sum_{q=1}^{N_q} \hat{w}_q \left\{ \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_q \hat{\phi}_j \Big|_{N_q - q + 1} \right\}.\end{aligned}$$

Now for the term $\langle \partial_n v^-, u^+ \rangle_e$

$$\begin{aligned}\langle \partial_n v^-, u^+ \rangle_e &= \left\langle \nabla v^- \cdot n, \sum_{j=1}^{N} \alpha_j^+ \phi_j^+ \right\rangle_e \\ &= \sum_{j=1}^{N} \alpha_j^+ \left\langle \sum_{i=1}^{N} \partial_n \phi_i^-, \phi_j^+ \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \partial_n \phi_i^-, \phi_j^+ \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}&\left\langle \partial_n \phi_i^-, \phi_j^+ \right\rangle_e \\ &\quad = \frac{h_e}{2|K^-|} \sum_{q=1}^{N_q} \hat{w}_q \left\{ \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_{N_q - q + 1} \hat{\phi}_j \Big|_q \right\}.\end{aligned}$$

It is important to note that for any interior edge e and all $i, j = 1$ to N

$$\left(\left\langle \partial_n \phi_i^-, \phi_j^+ \right\rangle_e \right) = \left(\left\langle \partial_n \phi_j^-, \phi_i^+ \right\rangle_e \right)^{\mathsf{T}}$$

and

$$\left(\left\langle \partial_n \phi_i^+, \phi_j^- \right\rangle_e \right) = \left(\left\langle \partial_n \phi_j^+, \phi_i^- \right\rangle_e \right)^{\mathsf{T}}$$

which implies that one has to calculate and store only one off-diagonal matrix block for each interior

edge e.

Therefore for the Baker formulation one obtains

$$A_{e,1}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \left\langle \partial_n \phi_j^+, \phi_i^- \right\rangle_e \right)_{i,j=1}^N \tag{A.10}$$

and

$$A_{e,1}^{(-,+)} = \left(A_{e,1}^{(+,-)} \right)^{\mathsf{T}} \tag{A.11}$$

thus giving

$$A_{e,1} = A_{e,1}^{(-,+)} + A_{e,1}^{(+,-)}. \tag{A.12}$$

Note that the superscripts $(-,+), (+,-)$ indicate that the matrix block is acting on the vector coefficients α^+, α^- respectively.

For the Arnold formulation

$$A_{e,1,+}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \frac{1}{2} \left(\left\langle \partial_n \phi_j^+, \phi_i^- \right\rangle_e - \left\langle \partial_n \phi_i^-, \phi_j^+ \right\rangle_e \right) \right)_{i,j=1}^N \tag{A.13}$$

$$A_{e,1,-}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \frac{1}{2} \left(\left\langle \partial_n \phi_i^+, \phi_j^- \right\rangle_e - \left\langle \partial_n \phi_j^-, \phi_i^+ \right\rangle_e \right) \right)_{i,j=1}^N \tag{A.14}$$

and

$$A_{e,1,+}^{(-,+)} = \left(A_{e,1,+}^{(+,-)} \right)^{\mathsf{T}} \tag{A.15}$$

$$A_{e,1,-}^{(-,+)} = \left(A_{e,1,-}^{(+,-)} \right)^{\mathsf{T}} \tag{A.16}$$

thus giving

$$A_{e,1,+} = A_{e,1,+}^{(-,+)} + A_{e,1,+}^{(+,-)} \tag{A.17}$$

$$A_{e,1,-} = A_{e,1,-}^{(-,+)} + A_{e,1,-}^{(+,-)} \tag{A.18}$$

and

$$A_{e,1} = A_{e,1,+} + A_{e,1,-} \tag{A.19}$$

Off-Diagonal Blocks: Penalty Mix Terms.

Penalty mixing terms $\gamma h_e^{-1}\langle u^+, v^-\rangle_e$ and $\gamma h_e^{-1}\langle u^-, v^+\rangle_e$ also contribute to formation of the off-diagonal blocks A_e. Following a similar path to what was done in the previous section

$$\gamma h_e^{-1}\langle u^+, v^-\rangle_e = \gamma h_e^{-1} \sum_{i,j=1}^{N} \alpha_j^+ \langle \phi_j^+, \phi_i^-\rangle_e$$

with

$$\langle \phi_j^+, \phi_i^-\rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_q \hat{\phi}_i \Big|_{N_q-q+1}$$

implies

$$\gamma h_e^{-1}\langle \phi_j^+, \phi_i^-\rangle_e = \gamma \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_q \hat{\phi}_i \Big|_{N_q-q+1}.$$

Note here that as before the h_e^{-1} weighting factor is absorbed into the edge quadrature. Similarly

$$\gamma h_e^{-1}\langle u^-, v^+\rangle_e = \gamma h_e^{-1} \sum_{i,j=1}^{N} \alpha_j^- \langle \phi_j^-, \phi_i^+\rangle_e$$

with

$$\langle \phi_j^-, \phi_i^+\rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_{N_q-q+1} \hat{\phi}_i \Big|_q$$

implies

$$\gamma h_e^{-1}\langle \phi_j^-, \phi_i^+\rangle_e = \gamma \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_{N_q-q+1} \hat{\phi}_i \Big|_q.$$

Therefore

$$\left(h_e^{-1}\langle \phi_j^-, \phi_i^+\rangle_e\right) = \left(h_e^{-1}\langle \phi_j^+, \phi_i^-\rangle_e\right)^{\mathsf{T}}$$

Thus, for both the Baker and Arnold formulations the penalty contributions to A_e can be represented as

$$A_{e,2}^{(-,+)} := \left(\sum_{e\in\mathcal{E}^I} \gamma h_e^{-1}\langle \phi_j^+, \phi_i^-\rangle_e \right)_{i,j=1}^{N} \tag{A.20}$$

Table A.1: Bilinear form calculational routines

Terms	Routines
$A_{K,1}$	`Diri`
$A_{K,2}, A^{+}_{K,2}$	`Flux_p`
$A^{-}_{K,2}$	`Flux_m`
$A^{+}_{K,3}, A^{-}_{K,3}$	`pnlt` template (`templs.c`)
$A^{(-,+)}_{e,1}$	`Flux_mix_p`
$A^{(-,+)}_{e,1,+}, A^{(-,+)}_{e,1,-}$	`Flux_mix_p, Flux_mix_m`
$A^{(-,+)}_{e,2}$	`pnltmx` template (`templs.c`)

and

$$A^{(+,-)}_{e,2} = \left(\sum_{e \in \mathcal{E}^I} \gamma h_e^{-1} \langle \phi_j^-, \phi_i^+ \rangle_e \right)_{i,j=1}^{N} \tag{A.21}$$

implies

$$A_{e,2} = A^{(-,+)}_{e,2} + A^{(+,-)}_{e,2}. \tag{A.22}$$

Off-Diagonal Block Assembly.

For both the Baker and Arnold formulations, for all $e \in \mathcal{E}^I$, assembly of A_e can be described as

$$A_e = A_{e,1} - A_{e,2}. \tag{A.23}$$

Stiffness Matrix Assembly.

To put this all together then, the stiffness matrix A for the Baker formulation can be assembled as:

$$A = \sum_{K \in \mathcal{T}_h} A_K + \sum_{e \in \mathcal{E}^I} A_e \tag{A.24}$$

Note that A_K also involves summing over edges e, however this decomposition of the matrix clearly delineates the diagonal blocks from the off-diagonal blocks. Routines used to calculate specific terms in the bilinear form are listed in Table A.1.

Appendix B

Ell2 Test Problems

Test Problem - f0

Domain Ω: Figure B.1

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = 1 & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = 1$.
This problem is trivial and has a constant solution providing a quick check on overall code functionality.

Test Problem - f1

Domain Ω: Figure B.1

$$\begin{cases} -\Delta u = -4 & \text{in } \Omega \\ u = x^2 + y^2 & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = x^2 + y^2$.
This problem has a smooth polynomial solution of degree 2. When elements involving polynomials of degree 2 or greater are used, the code should produce exact solutions from the initial mesh.

Test Problem - f2

Domain Ω: Figure B.1

$$\begin{cases} -\Delta u = 2x(1-x) + 2y(1-y) & \text{in } \Omega \\ u = 0 & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = xy(1-x)(1-y)$.
This problem has a smooth polynomial solution of degree 4. When elements involving polynomials of degree 4 are used, the code should produce exact solutions from the initial mesh. The problem has homogeneous Dirichlet boundary conditions and the solution is solely driven by the forcing function.

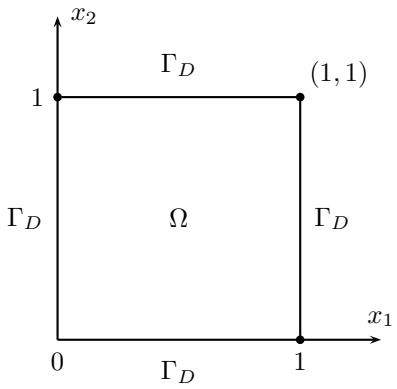


Figure B.1: Square Domain

Test Problem - f3

Domain Ω: Figure B.1

$$\begin{cases} -\Delta u = 2\pi^2 \sin(\pi x)\sin(\pi y) & \text{in } \Omega \\ u = 0 & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = \sin(\pi x)\sin(\pi y)$.
This problem has a smooth non-polynomial solution. The problem has homogeneous Dirichlet boundary conditions and the solution is solely driven by the forcing function.

Test Problem - f4

Domain Ω: Figure B.1

$$\begin{cases} -\Delta u = 128\pi^2 \sin(8\pi x)\sin(8\pi y) & \text{in } \Omega \\ u = 0 & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = \sin(8\pi x)\sin(8\pi y)$.
This problem has a smooth non-polynomial solution which is oscillatory across the domain. The problem has homogeneous Dirichlet boundary conditions and the solution is solely driven by the forcing function.

Test Problem - f5

Domain Ω: Figure B.1

$$\begin{cases} -\Delta u = 2\pi^2(\sin(\pi x)\sin(\pi y) - 1) & \text{in } \Omega \\ u = 1 & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = \sin(\pi x)\sin(\pi y) + 1$.
This problem has a smooth non-polynomial solution, similar to test problem f3, shifted up by 1. This problem tests more aggressively the application of non-homogeneous Dirichlet boundary conditions, with the solution being driven by both the forcing function and the BC's.

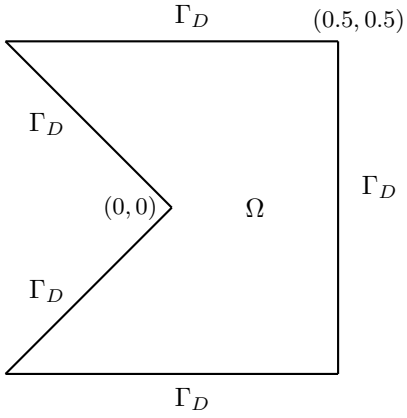


Figure B.2: Notch Domain

Test Problem - f6

Domain Ω: Figure B.2

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = r^{2/3}\sin(2/3\theta) & \text{on } \Gamma_D \end{cases}$$

Exact solution: $u = r^{2/3}\sin(2/3\theta)$.
This problem has a point singularity in the first derivative at the origin. This problem really stresses how well the adaptive algorithms work. Note also that the solution to this problem is solely driven by the trace of the solution on the boundary.

Test Problem - f7

Domain Ω: Figure B.3

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = 0 & \text{on } \Gamma_D \\ \dfrac{\partial u}{\partial n} = -1 & \text{on } \Gamma_N \end{cases}$$

Exact solution: Not known.
This problem has two point singularities due to mismatch in the first derivative at the corners where $\Gamma_D \cap \Gamma_N \neq \emptyset$. This problem tests to see if mixed boundary conditions and adaptive strategies are working properly.

Test Problem - f8

Domain Ω: Figure B.3

$$\begin{cases} -\Delta u = 2\pi^2 \sin(\pi x)\sin(\pi y) & \text{in } \Omega \\ u = 0 & \text{on } \Gamma_D \\ \dfrac{\partial u}{\partial n} = -\pi \sin(\pi y) & \text{on } \Gamma_N \end{cases}$$

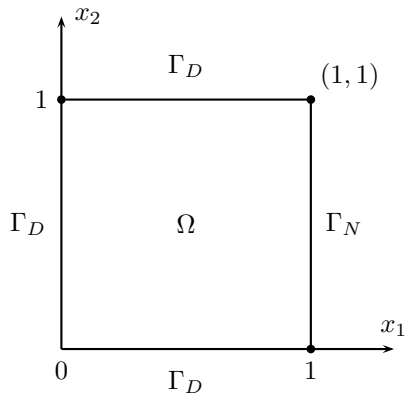


Figure B.3: Mixbc Domain

Exact solution: $u = \sin(\pi x)\sin(\pi y)$.
This problem has a smooth non-polynomial solution and is the same as the solution as for test problem f3. The problem has homogeneous Dirichlet boundary conditions and non-homogeneous Neumann boundary conditions. The solution is driven by the forcing function and the boundary conditions.

Appendix C

`Ell4` Stiffness Matrix Assembly Detail

Diagonal Blocks: Laplacian term.

Assume that on K the following functions are defined

$$u(\mathbf{x}) = \sum_{j=1}^{N} u(\mathbf{x}_j^K)\phi_j^K(\mathbf{x})$$

and

$$v(\mathbf{x}) = \sum_{i=1}^{N} \phi_i^K(\mathbf{x})$$

where N is the number of degrees of freedom and for convenience we will drop the superscript K annotation unless otherwise needed. Then

$$\begin{aligned}
\left(\Delta u(\mathbf{x}), \Delta v(\mathbf{x})\right)_K &= \int_K \Delta u(\mathbf{x})\Delta v(\mathbf{x})\, d\mathbf{x} \\
&= \int_K \Delta u(\mathbf{x})\Delta v(\mathbf{x})\, d\mathbf{x} \\
&= \sum_{i,j=1}^{N} u(\mathbf{x}_j^K) \int_K \Delta\phi_j \Delta\phi_i \, d\mathbf{x} \\
&= \sum_{i,j=1}^{N} \left(\Delta\phi_j, \Delta\phi_i\right)_K u(\mathbf{x}_j^K) = \sum_{i,j=1}^{N} \left(\Delta\phi_j, \Delta\phi_i\right)_K \alpha_j^K.
\end{aligned}$$

where $\{\alpha_j^K\}_{j=1}^N$ are the unknowns (dofs) for triangle K. Applying change of variable transformations for quadrature over the reference element $\hat{K}$ implies

$$\begin{aligned}\Delta\phi_j\Delta\phi_i &= \left(\frac{1}{2|K|}\right)^2\left[c_1\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right]\left(\frac{1}{2|K|}\right)^2\left[c_1\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_i}{\partial\hat{y}^2}\right]\\ &= \left(\frac{1}{2|K|}\right)^4\left[c_1\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right]\left[c_1\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_i}{\partial\hat{y}^2}\right]\end{aligned}$$

Thus, with N_q being the number of quadrature points for a chosen quadrature rule and evaluating each partial derivative at quadrature points q, $q=1,\ldots,N_q$ with quadrature weight $\hat{w}_q$,

$$\begin{aligned}&(\Delta\phi_j,\Delta\phi_i)_K = \\ &\quad\left(\frac{1}{2|K|}\right)^4\left[(2|K|)\sum_{q=1}^{N_q}\hat{w}_q\left(c_1\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right)_q\left(c_1\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_i}{\partial\hat{y}^2}\right)_q\right]\end{aligned}$$

or

$$\begin{aligned}&(\Delta\phi_j,\Delta\phi_i)_K = \\ &\qquad\left(\frac{1}{2|K|}\right)^3\left[\sum_{q=1}^{N_q}\hat{w}_q\left(c_1\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_j}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_j}{\partial\hat{y}^2}\right)_q\left(c_1\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}^2}+c_2\frac{\partial^2\hat{\phi}_i}{\partial\hat{x}\partial\hat{y}}+c_3\frac{\partial^2\hat{\phi}_i}{\partial\hat{y}^2}\right)_q\right]\end{aligned}$$

Note that

$$A_{K,1} := \Big((\Delta\phi_j,\Delta\phi_i)_K\Big)_{i,j=1}^N \tag{C.1}$$

is a symmetric matrix and is the *Laplacian* portion of the diagonal block matrix A_K.

Diagonal Blocks: Flux terms.

Associated with each interior edge $e\in\mathcal{E}^I$ are two triangles, (K^+,K^-). The edge integrals which contribute to the diagonal blocks relating to fluxes are

$$\begin{aligned}&\left\langle\partial_n(\Delta v^+),u^+\right\rangle_e-\left\langle\Delta v^+,\partial_n u^+\right\rangle_e+\left\langle\partial_n(\Delta u^+),v^+\right\rangle_e-\left\langle\Delta u^+,\partial_n v^+\right\rangle_e,\\ &\left\langle\partial_n(\Delta v^-),u^-\right\rangle_e-\left\langle\Delta v^-,\partial_n u^-\right\rangle_e+\left\langle\partial_n(\Delta u^-),v^-\right\rangle_e-\left\langle\Delta u^-,\partial_n v^-\right\rangle_e\end{aligned}$$

for K^+ and K^-, respectively, and where n is the unit outward normal to K^+.

First consider the term $\langle \partial_n(\Delta v^+), u^+ \rangle_e$:

$$\begin{aligned}\langle \partial_n(\Delta v^+), u^+ \rangle_e &= \left\langle \sum_{j=1}^{N} \alpha_j^+ \phi_j^+, \partial_n(\Delta v^+) \right\rangle_e \\ &= \sum_{j=1}^{N} \alpha_j^+ \left\langle \phi_j^+, \sum_{i=1}^{N} \partial_n(\Delta \phi_i^+) \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \phi_j^+, \partial_n(\Delta \phi_i^+) \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \phi_j^+, \partial_n(\Delta \phi_i^+) \right\rangle_e = h_e \left(\frac{1}{2|K|}\right)^3 \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \hat{\phi}_j \Bigg[\left(d_{11} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{x}^3} + d_{12} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{x}^2 \partial \hat{y}} + d_{13} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{x} \partial \hat{y}^2} + d_{14} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{y}^3} \right) n_x \\ + \left(d_{21} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{x}^3} + d_{22} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{x}^2 \partial \hat{y}} + d_{23} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{x} \partial \hat{y}^2} + d_{24} \frac{\partial^3 \hat{\phi}_i}{\partial \hat{y}^3} \right) n_y \Bigg] \Bigg\}_q .\end{aligned}$$

Now consider the term $\langle \partial_n(\Delta u^+), v^+ \rangle_e$:

$$\begin{aligned}\langle \partial_n(\Delta u^+), v^+ \rangle_e &= \left\langle \sum_{i=1}^{N} \phi_i^+, \partial_n(\Delta u^+) \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \phi_i^+, \sum_{j=1}^{N} \alpha_j^+ \partial_n(\Delta \phi_j^+) \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \phi_i^+, \partial_n(\Delta \phi_j^+) \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \phi_i^+, \partial_n(\Delta \phi_j^+) \right\rangle_e = h_e \left(\frac{1}{2|K|}\right)^3 \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \hat{\phi}_i \Bigg[\left(d_{11} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^3} + d_{12} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^2 \partial \hat{y}} + d_{13} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}^2} + d_{14} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{y}^3} \right) n_x \\ + \left(d_{21} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^3} + d_{22} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^2 \partial \hat{y}} + d_{23} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}^2} + d_{24} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{y}^3} \right) n_y \Bigg] \Bigg\}_q .\end{aligned}$$

Note that similar formulae exist for the terms $\langle \partial_n(\Delta v^-), u^- \rangle_e$, $\langle \partial_n(\Delta u^-), v^- \rangle_e$.

Now consider the term $\langle \Delta v^+, \partial_n u^+ \rangle_e$:

$$\begin{aligned}\langle \Delta v^+, \partial_n u^+ \rangle_e &= \left\langle \sum_{i=1}^{N} \Delta\phi_i^+, \nabla u^+ \cdot n \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \Delta\phi_i^+, \sum_{j=1}^{N} \alpha_j^+ \left(\frac{\partial \phi_j^+}{\partial x} n_x + \frac{\partial \phi_j^+}{\partial y} n_y \right) \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \Delta\phi_i^+, \partial_n \phi_j^+ \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \Delta\phi_i^+, \partial_n \phi_j^+ \right\rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left(\frac{1}{2|K|} \right)^2 \left[c_1 \frac{\partial^2 \hat{\phi}_i}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_i}{\partial \hat{x} \partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_i}{\partial \hat{y}^2} \right] \\ \cdot \frac{1}{2|K|} \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right] \Bigg\}_q\end{aligned}$$

or

$$\begin{aligned}\left\langle \Delta\phi_i^+, \partial_n \phi_j^+ \right\rangle_e = h_e \left(\frac{1}{2|K|} \right)^3 \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left[c_1 \frac{\partial^2 \hat{\phi}_i}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_i}{\partial \hat{x} \partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_i}{\partial \hat{y}^2} \right] \\ \cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right] \Bigg\}_q .\end{aligned}$$

Now consider the term $\langle \Delta u^+, \partial_n v^+ \rangle_e$:

$$\begin{aligned}\langle \Delta u^+, \partial_n v^+ \rangle_e &= \left\langle \sum_{j=1}^{N} \alpha_j^+ \Delta\phi_j^+, \nabla v^+ \cdot n \right\rangle_e \\ &= \sum_{j=1}^{N} \alpha_j^+ \left\langle \Delta\phi_j^+, \sum_{i=1}^{N} \partial_n \phi_i^+ \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \Delta\phi_j^+, \partial_n \phi_i^+ \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \Delta\phi_j^+, \partial_n \phi_i^+ \right\rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left(\frac{1}{2|K|} \right)^2 \left[c_1 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{y}^2} \right] \\ \cdot \frac{1}{2|K|} \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right] \Bigg\}_q\end{aligned}$$

or

$$\left\langle \Delta\phi_j^+, \partial_n\phi_i^+ \right\rangle_e = h_e \left(\frac{1}{2|K|}\right)^3 \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left[c_1 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x}\partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{y}^2} \right] \cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right] \Bigg\}_q .$$

As before, note that similar formulae exist for the terms $\langle \Delta v^-, \partial_n u^- \rangle_e$, $\langle \Delta u^-, \partial_n v^- \rangle_e$.

For the Baker formulation (Eq. 4.7) one obtains

$$A_{K,2} := \left(\sum_{\substack{e \in \partial K \\ K = K^+}} \left[\left\langle \phi_i^+, \partial_n(\Delta\phi_j^+) \right\rangle_e - \left\langle \Delta\phi_i^+, \partial_n\phi_j^+ \right\rangle_e + \left\langle \phi_j^+, \partial_n(\Delta\phi_i^+) \right\rangle_e - \left\langle \Delta\phi_j^+, \partial_n\phi_i^+ \right\rangle_e \right] \right)_{i,j=1}^{N} \tag{C.2}$$

is a symmetric matrix and contributes to the *Flux* portion of the diagonal block matrix A_K. For the Arnold formulation (Eq. 4.8)

$$A_{K,2}^+ := \left(\frac{1}{2} \sum_{\substack{e \in \partial K \\ K = K^+}} \left[\left\langle \phi_i^+, \partial_n(\Delta\phi_j^+) \right\rangle_e - \left\langle \Delta\phi_i^+, \partial_n\phi_j^+ \right\rangle_e + \left\langle \phi_j^+, \partial_n(\Delta\phi_i^+) \right\rangle_e - \left\langle \Delta\phi_j^+, \partial_n\phi_i^+ \right\rangle_e \right] \right)_{i,j=1}^{N} \tag{C.3}$$

and

$$A_{K,2}^- := \left(\frac{1}{2} \sum_{\substack{e \in \partial K \\ K = K^-}} \left[\left\langle \phi_i^-, \partial_n(\Delta\phi_j^-) \right\rangle_e - \left\langle \Delta\phi_i^-, \partial_n\phi_j^- \right\rangle_e + \left\langle \phi_j^-, \partial_n(\Delta\phi_i^-) \right\rangle_e - \left\langle \Delta\phi_j^-, \partial_n\phi_i^- \right\rangle_e \right] \right)_{i,j=1}^{N} \tag{C.4}$$

both contribute to A_K.

Diagonal Blocks: Penalty terms.

Penalty terms $\gamma h_e^{-1} \langle \partial_n u^+, \partial_n v^+ \rangle_e$, $\gamma h_e^{-1} \langle \partial_n u^-, \partial_n v^- \rangle_e$, $\gamma h_e^{-3} \langle u^+, v^+ \rangle_e$, and $\gamma h_e^{-3} \langle u^-, v^- \rangle_e$ also contribute to the formulation of the diagonal block A_K.

Following a similar path to what was done in the previous section

$$\gamma h_e^{-3} \langle u^+, v^+ \rangle_e = \gamma h_e^{-3} \sum_{i,j=1}^{N} \alpha_j^+ \langle \phi_j^+, \phi_i^+ \rangle_e$$

with

$$\langle \phi_j^+, \phi_i^+ \rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q [\hat{\phi}_j \hat{\phi}_i]_q$$

implies

$$\gamma h_e^{-3}\langle \phi_j^+, \phi_i^+\rangle_e = \gamma h_e^{-2}\sum_{q=1}^{N_q} \hat{w}_q[\hat{\phi}_j\hat{\phi}_i]_q.$$

Similarly for K^-

$$\gamma h_e^{-3}\langle \phi_j^-, \phi_i^-\rangle_e = \gamma h_e^{-2}\sum_{q=1}^{N_q} \hat{w}_q[\hat{\phi}_j\hat{\phi}_i]_q.$$

Derivative penalty terms require special consideration:

$$\gamma h_e^{-1}\langle \partial_n u^+, \partial_n v^+\rangle_e = \gamma h_e^{-1}\sum_{i,j=1}^{N} \alpha_j^+ \left\langle \partial_n\phi_j^+, \partial_n\phi_i^+\right\rangle_e$$

with

$$\begin{aligned}\left\langle \partial_n\phi_j^+, \partial_n\phi_i^+\right\rangle_e = h_e\sum_{q=1}^{N_q}\hat{w}_q\frac{1}{2|K|}\Bigg[\left(\hat{a}_{11}\frac{\partial\hat{\phi}_j}{\partial\hat{x}}+\hat{a}_{21}\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\right)n_x \\ +\left(\hat{a}_{12}\frac{\partial\hat{\phi}_j}{\partial\hat{x}}+\hat{a}_{22}\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\right)n_y\Bigg]_q\frac{1}{2|K|}\Bigg[\left(\hat{a}_{11}\frac{\partial\hat{\phi}_i}{\partial\hat{x}}+\hat{a}_{21}\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\right)n_x+\left(\hat{a}_{12}\frac{\partial\hat{\phi}_i}{\partial\hat{x}}+\hat{a}_{22}\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\right)n_y\Bigg]_q\end{aligned}$$

implies

$$\begin{aligned}\gamma h_e^{-1}\left\langle \partial_n\phi_j^+, \partial_n\phi_i^-\right\rangle_e = \frac{\gamma}{(2|K|)^2}\sum_{q=1}^{N_q}\hat{w}_q\Bigg[\left(\hat{a}_{11}\frac{\partial\hat{\phi}_j}{\partial\hat{x}}+\hat{a}_{21}\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\right)n_x \\ +\left(\hat{a}_{12}\frac{\partial\hat{\phi}_j}{\partial\hat{x}}+\hat{a}_{22}\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\right)n_y\Bigg]_q\Bigg[\left(\hat{a}_{11}\frac{\partial\hat{\phi}_i}{\partial\hat{x}}+\hat{a}_{21}\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\right)n_x+\left(\hat{a}_{12}\frac{\partial\hat{\phi}_i}{\partial\hat{x}}+\hat{a}_{22}\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\right)n_y\Bigg]_q.\end{aligned}$$

Similarly for K^-

$$\begin{aligned}\gamma h_e^{-1}\left\langle \partial_n\phi_j^-, \partial_n\phi_i^-\right\rangle_e = \frac{\gamma}{(2|K|)^2}\sum_{q=1}^{N_q}\hat{w}_q\Bigg[\left(\hat{a}_{11}\frac{\partial\hat{\phi}_j}{\partial\hat{x}}+\hat{a}_{21}\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\right)n_x \\ +\left(\hat{a}_{12}\frac{\partial\hat{\phi}_j}{\partial\hat{x}}+\hat{a}_{22}\frac{\partial\hat{\phi}_j}{\partial\hat{y}}\right)n_y\Bigg]_q\Bigg[\left(\hat{a}_{11}\frac{\partial\hat{\phi}_i}{\partial\hat{x}}+\hat{a}_{21}\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\right)n_x+\left(\hat{a}_{12}\frac{\partial\hat{\phi}_i}{\partial\hat{x}}+\hat{a}_{22}\frac{\partial\hat{\phi}_i}{\partial\hat{y}}\right)n_y\Bigg]_q.\end{aligned}$$

Therefore, the penalty contributions to A_K can be represented as

$$A_{K,3}^+ := \left(\sum_{\substack{e\in\partial K\\ K=K^+\\ e\in\mathcal{E}}}\left[\gamma h_e^{-3}\langle\phi_j^+,\phi_i^+\rangle_e+\gamma h_e^{-1}\left\langle\partial_n\phi_j^+,\partial_n\phi_i^+\right\rangle_e\right]\right)_{i,j=1}^{N} \tag{C.5}$$

and

$$A^-_{K,3} := \left(\sum_{\substack{e\in\partial K \\ K=K^- \\ e\in\mathcal{E}^I}} \left[\gamma h_e^{-1} \langle \phi_j^-, \phi_i^- \rangle_e + \gamma h_e^{-1} \left\langle \partial_n \phi_j^-, \partial_n \phi_i^- \right\rangle_e \right] \right)_{i,j=1}^N . \tag{C.6}$$

Diagonal Block Assembly.

For the Baker formulation, K^- contributions to A_K only occur through the penalty terms $A^-_{K,3}$. For the Arnold formulation, contributions to A_K involve the flux terms $A^-_{K,2}$ and the penalty terms $A^-_{K,3}$. Since DG allows the presence of hanging nodes, special care must be taken with respect to accumulation of A_K components due to K^- contributions. In the case where a hanging node is present along an edge $e \in \partial K$ and $K = K^-$, then $A^-_{K,3}$ terms will accumulate in A_K from all (up to two) triangles adjacent to K along e. Therefore, $A^-_{K,3}$ can be rewritten to take hanging nodes into account as

$$\begin{aligned} A^-_{K,3} := \Bigg(\sum_{\substack{e\in\partial K \\ K=K^- \\ e\in\mathcal{E}^I}} \sigma_e^K \left[\gamma h_e^{-3} \langle \phi_j^-, \phi_i^- \rangle_e + \gamma h_e^{-1} \left\langle \partial_n \phi_j^-, \partial_n \phi_i^- \right\rangle_e \right] \\ + \sum_{\substack{e\in\partial K \\ K=K^- \\ e\notin\mathcal{E}^I}} \Big[\gamma h_{e_1}^{-3} \langle \phi_j^-, \phi_i^- \rangle_{e_1} + \gamma h_{e_2}^{-3} \langle \phi_j^-, \phi_i^- \rangle_{e_2} \\ + \gamma h_{e_1}^{-1} \left\langle \partial_n \phi_j^-, \partial_n \phi_i^- \right\rangle_{e_1} + \gamma h_{e_2}^{-1} \left\langle \partial_n \phi_j^-, \partial_n \phi_i^- \right\rangle_{e_2} \Big] \Bigg)_{i,j=1}^N . \end{aligned} \tag{C.7}$$

Note that we assume here that if $K = K^-$ and along edge $e \in \partial K$ there are two neighboring triangles (i.e., edge e is split into two edges e_1, e_2), then both neighboring triangles K_{e_1}, K_{e_2} are *both* considered to be $K^+_{e_1}, K^+_{e_2}$ relative to e_1, e_2 respectively.

Then for all $K \in \mathcal{T}_h$, assembly of A_K can be described for the Baker formulation as

$$A_K = A_{K,1} + A_{K,2} + A^+_{K,3} + A^-_{K,3} \tag{C.8}$$

and for the Arnold formulation as

$$A_K = A_{K,1} + A^+_{K,2} + A^-_{K,2} + A^+_{K,3} + A^-_{K,3}. \tag{C.9}$$

Off-Diagonal Blocks: Flux Mix Terms.

The edge integrals which contribute to the off-diagonal blocks relating to fluxes are

- $\langle \partial_n(\Delta u^+), v^- \rangle_e$, $\langle \Delta u^+, \partial_n v^- \rangle_e$,
- $\langle \partial_n(\Delta v^+), u^- \rangle_e$,$\langle \Delta v^+, \partial_n u^- \rangle_e$,
- $\langle \partial_n(\Delta u^-), v^+ \rangle_e$, $\langle \Delta u^-, \partial_n v^+ \rangle_e$
- $\langle \partial_n(\Delta v^-), u^+ \rangle_e$,$\langle \Delta v^-, \partial_n u^+ \rangle_e$

which describe the *mixing* of fluxes through edge e.

Now consider the term $\langle \partial_n(\Delta u^+), v^- \rangle_e$:

$$\begin{aligned}\langle \partial_n(\Delta u^+), v^- \rangle_e &= \left\langle \nabla(\Delta u^+) \cdot n, \sum_{i=1}^{N} \phi_i^- \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \sum_{j=1}^{N} \alpha_j^+ \partial_n(\Delta \phi_j^+), \phi_i^- \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \partial_n(\Delta \phi_j^+), \phi_i^- \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \partial_n(\Delta \phi_j^+), \phi_i^- \right\rangle_e = h_e \left(\frac{1}{2|K^+|}\right)^3 \sum_{q=1}^{N_q} \hat{w}_q \Bigg[\left(d_{11} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^3} + d_{12} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^2 \partial \hat{y}} + d_{13} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}^2} + d_{14} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{y}^3} \right) n_x \\ + \left(d_{21} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^3} + d_{22} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^2 \partial \hat{y}} + d_{23} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}^2} + d_{24} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{y}^3} \right) n_y \Bigg]_q \hat{\phi}_i \Big|_{N_q - q + 1}\end{aligned}$$

Now consider the term $\langle \partial_n(\Delta u^-), v^+ \rangle_e$:

$$\begin{aligned}\langle \partial_n(\Delta u^-), v^+ \rangle_e &= \left\langle \nabla(\Delta u^-) \cdot n, \sum_{i=1}^{N} \phi_i^+ \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \sum_{j=1}^{N} \alpha_j^- \partial_n(\Delta \phi_j^-), \phi_i^- \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^- \left\langle \partial_n(\Delta \phi_j^-), \phi_i^- \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \partial_n(\Delta \phi_j^-), \phi_i^+ \right\rangle_e = h_e \left(\frac{1}{2|K^-|}\right)^3 \sum_{q=1}^{N_q} \hat{w}_q \Bigg[\left(d_{11} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^3} + d_{12} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^2 \partial \hat{y}} + d_{13} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}^2} + d_{14} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{y}^3} \right) n_x \\ + \left(d_{21} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^3} + d_{22} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x}^2 \partial \hat{y}} + d_{23} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}^2} + d_{24} \frac{\partial^3 \hat{\phi}_j}{\partial \hat{y}^3} \right) n_y \Bigg]_{N_q - q + 1} \hat{\phi}_i \Big|_q\end{aligned}$$

Note that similar formulae exist for $\langle \partial_n(\Delta v^+), u^- \rangle_e$ and $\langle \partial_n(\Delta v^-), u^+ \rangle_e$.

Now for the term $\langle \Delta u^+, \partial_n v^- \rangle_e$

$$\begin{aligned}\langle \Delta u^+, \partial_n v^- \rangle_e &= \left\langle \Delta u^+, \sum_{i=1}^{N} \partial_n \phi_i^- \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \sum_{j=1}^{N} \alpha_j^+ \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^+ \left\langle \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\begin{aligned}\left\langle \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left(\frac{1}{2|K^+|} \right)^2 \left[c_1 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{y}^2} \right]_q \\ \cdot \frac{1}{2|K^-|} \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_{N_q - q + 1} \Bigg\}.\end{aligned}$$

or

$$\begin{aligned}\left\langle \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e = \frac{h_e}{8|K^+|^2|K^-|} \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left[c_1 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{y}^2} \right]_q \\ \cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_{N_q - q + 1} \Bigg\}.\end{aligned}$$

Now for the term $\langle \Delta u^-, \partial_n v^+ \rangle_e$

$$\begin{aligned}\langle \Delta u^-, \partial_n v^+ \rangle_e &= \left\langle \Delta u^-, \sum_{i=1}^{N} \partial_n \phi_i^+ \right\rangle_e \\ &= \sum_{i=1}^{N} \left\langle \sum_{j=1}^{N} \alpha_j^- \Delta \phi_j^- , \partial_n \phi_i^+ \right\rangle_e \\ &= \sum_{i,j=1}^{N} \alpha_j^- \left\langle \Delta \phi_j^-, \partial_n \phi_i^+ \right\rangle_e .\end{aligned}$$

Applying quadrature rule implies that

$$\left\langle \Delta\phi_j^-, \partial_n \phi_i^+ \right\rangle_e = \frac{h_e}{8|K^-|^2|K^+|} \sum_{q=1}^{N_q} \hat{w}_q \Bigg\{ \left[c_1 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x}^2} + c_2 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{x} \partial \hat{y}} + c_3 \frac{\partial^2 \hat{\phi}_j}{\partial \hat{y}^2} \right]_{N_q - q + 1} \cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_q \Bigg\}.$$

Note that similar expressions exist for $\langle \Delta v^+, \partial_n u^- \rangle_e$ and $\langle \Delta v^-, \partial_n u^+ \rangle_e$.

It is important to note that for any interior edge e and all $i, j = 1$ to N

$$\left(\left\langle \partial_n (\Delta \phi_j^+), \phi_i^- \right\rangle_e \right) = \left(\left\langle \partial_n (\Delta \phi_i^+), \phi_j^- \right\rangle_e \right)^{\mathsf{T}}$$

and

$$\left(\left\langle \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e \right) = \left(\left\langle \Delta \phi_i^+, \partial_n \phi_j^- \right\rangle_e \right)^{\mathsf{T}}$$

which implies that one has to calculate and store only one off-diagonal matrix block for each interior edge e.

Therefore for the Baker formulation one obtains

$$A_{e,1}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \left\langle \partial_n (\Delta \phi_j^+), \phi_i^- \right\rangle_e - \left\langle \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e \right)_{i,j=1}^{N} \tag{C.10}$$

and

$$A_{e,1}^{(-,+)} = \left(A_{e,1}^{(+,-)} \right)^{\mathsf{T}} \tag{C.11}$$

thus giving

$$A_{e,1} = A_{e,1}^{(-,+)} + A_{e,1}^{(+,-)}. \tag{C.12}$$

Note that the superscripts $(-,+), (+,-)$ indicate that the matrix block is acting on the vector coefficients α^+, α^- respectively.

For the Arnold formulation

$$A_{e,1,+}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \frac{1}{2} \left(\left\langle \partial_n (\Delta \phi_j^+), \phi_i^- \right\rangle_e - \left\langle \Delta \phi_j^+, \partial_n \phi_i^- \right\rangle_e - \left\langle \partial_n (\Delta \phi_i^-), \phi_j^+ \right\rangle_e + \left\langle \Delta \phi_i^-, \partial_n \phi_j^+ \right\rangle_e \right) \right)_{i,j=1}^{N} \tag{C.13}$$

$$A_{e,1,-}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \frac{1}{2} \left(\left\langle \partial_n (\Delta \phi_i^+), \phi_j^- \right\rangle_e - \left\langle \Delta \phi_i^+, \partial_n \phi_j^- \right\rangle_e - \left\langle \partial_n (\Delta \phi_j^-), \phi_i^+ \right\rangle_e + \left\langle \Delta \phi_j^-, \partial_n \phi_i^+ \right\rangle_e \right) \right)_{i,j=1}^{N} \tag{C.14}$$

and

$$A_{e,1,+}^{(-,+)} = \left(A_{e,1,+}^{(+,-)}\right)^{\mathsf{T}} \tag{C.15}$$

$$A_{e,1,-}^{(-,+)} = \left(A_{e,1,-}^{(+,-)}\right)^{\mathsf{T}} \tag{C.16}$$

thus giving

$$A_{e,1,+} = A_{e,1,+}^{(-,+)} + A_{e,1,+}^{(+,-)} \tag{C.17}$$

$$A_{e,1,-} = A_{e,1,-}^{(-,+)} + A_{e,1,-}^{(+,-)} \tag{C.18}$$

and

$$A_{e,1} = A_{e,1,+} + A_{e,1,-} \tag{C.19}$$

Off-Diagonal Blocks: Penalty Mix Terms.

Penalty mixing terms

- $\gamma h_e^{-3}\langle \phi_j^+, \phi_i^- \rangle_e$
- $\gamma h_e^{-3}\langle \phi_j^-, \phi_i^+ \rangle_e$
- $\gamma h_e^{-1}\left\langle \partial_n\phi_j^+, \partial_n\phi_i^- \right\rangle_e$
- $\gamma h_e^{-1}\left\langle \partial_n\phi_j^-, \partial_n\phi_i^+ \right\rangle_e$

also contribute to formation of the off-diagonal blocks A_e. Following a similar path to what was done in the previous section

$$\gamma h_e^{-3}\langle u^+, v^- \rangle_e = \gamma h_e^{-3} \sum_{i,j=1}^{N} \alpha_j^+ \langle \phi_j^+, \phi_i^- \rangle_e$$

with

$$\langle \phi_j^+, \phi_i^- \rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_q \hat{\phi}_i \Big|_{N_q-q+1}$$

implies

$$\gamma h_e^{-3}\langle \phi_j^+, \phi_i^- \rangle_e = \gamma h_e^{-2} \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_q \hat{\phi}_i \Big|_{N_q-q+1}.$$

Similarly

$$\gamma h_e^{-3}\langle u^-, v^+ \rangle_e = \gamma h_e^{-3} \sum_{i,j=1}^{N} \alpha_j^- \langle \phi_j^-, \phi_i^+ \rangle_e$$

with

$$\langle \phi_j^-, \phi_i^+ \rangle_e = h_e \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_{N_q-q+1} \hat{\phi}_i \Big|_q$$

implies

$$\gamma h_e^{-3} \langle \phi_j^-, \phi_i^+ \rangle_e = \gamma h_e^{-2} \sum_{q=1}^{N_q} \hat{w}_q \hat{\phi}_j \Big|_{N_q-q+1} \hat{\phi}_i \Big|_q .$$

Therefore

$$\left(h_e^{-3} \langle \phi_j^-, \phi_i^+ \rangle_e\right) = \left(h_e^{-3} \langle \phi_j^+, \phi_i^- \rangle_e\right)^{\mathsf{T}}$$

Now for the first derivative penalty terms

$$\gamma h_e^{-1} \langle \partial_n u^+, \partial_n v^- \rangle_e = \gamma h_e^{-1} \sum_{i,j=1}^{N} \alpha_j^+ \langle \partial_n \phi_j^+, \partial_n \phi_i^- \rangle_e$$

with

$$\left\langle \partial_n \phi_j^+, \partial_n \phi_i^- \right\rangle_e = \frac{1}{4|K^+||K^-|} \sum_{q=1}^{N_q} \hat{w}_q \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right]_q$$
$$\cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_{N_q-q+1} .$$

implies

$$\gamma h_e^{-1} \left\langle \partial_n \phi_j^+, \partial_n \phi_i^- \right\rangle_e = \frac{\gamma}{4|K^+||K^-|} \sum_{q=1}^{N_q} \hat{w}_q \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right]_q$$
$$\cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_{N_q-q+1} .$$

Similarly

$$\gamma h_e^{-1} \langle \partial_n u^-, \partial_n v^+ \rangle_e = \gamma h_e^{-1} \sum_{i,j=1}^{N} \alpha_j^- \langle \partial_n \phi_j^-, \partial_n \phi_i^+ \rangle_e$$

with

$$\left\langle \partial_n \phi_j^-, \partial_n \phi_i^+ \right\rangle_e = \frac{1}{4|K^+||K^-|} \sum_{q=1}^{N_q} \hat{w}_q \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x \right.$$
$$\left. + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right]_{N_q - q + 1} \cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_q .$$

implies

$$\gamma h_e^{-1} \left\langle \partial_n \phi_j^-, \partial_n \phi_i^+ \right\rangle_e = \frac{\gamma}{4|K^+||K^-|} \sum_{q=1}^{N_q} \hat{w}_q \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_x \right.$$
$$\left. + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_j}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_j}{\partial \hat{y}} \right) n_y \right]_{N_q - q + 1} \cdot \left[\left(\hat{a}_{11} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{21} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_x + \left(\hat{a}_{12} \frac{\partial \hat{\phi}_i}{\partial \hat{x}} + \hat{a}_{22} \frac{\partial \hat{\phi}_i}{\partial \hat{y}} \right) n_y \right]_q .$$

Thus, for both the Baker and Arnold formulations the penalty contributions to A_e can be represented as

$$A_{e,2}^{(-,+)} := \left(\sum_{e \in \mathcal{E}^I} \gamma [h_e^{-3} \langle \phi_j^+, \phi_i^- \rangle_e + h_e^{-1} \langle \partial_n \phi_j^+, \partial_n \phi_i^- \rangle_e] \right)_{i,j=1}^N \tag{C.20}$$

and

$$A_{e,2}^{(+,-)} = \left(\sum_{e \in \mathcal{E}^I} \gamma [h_e^{-3} \langle \phi_j^-, \phi_i^+ \rangle_e + h_e^{-1} \langle \partial_n \phi_j^-, \partial_n \phi_i^+ \rangle_e] \right)_{i,j=1}^N \tag{C.21}$$

implies

$$A_{e,2} = A_{e,2}^{(-,+)} + A_{e,2}^{(+,-)}. \tag{C.22}$$

Off-Diagonal Block Assembly.

For both the Baker and Arnold formulations, for all $e \in \mathcal{E}^I$, assembly of A_e can be described as

$$A_e = -A_{e,1} - A_{e,2}. \tag{C.23}$$

Stiffness Matrix Assembly.

To put this all together then, the stiffness matrix A for the Baker formulation can be assembled as:

$$A = \sum_{K \in \mathcal{T}_h} A_K + \sum_{e \in \mathcal{E}^I} A_e \tag{C.24}$$

Note that A_K also involves summing over edges e, however this decomposition of the matrix clearly delineates the diagonal blocks from the off-diagonal blocks. Routines used to calculate specific terms in the bilinear form are listed in Table XXXX.

Appendix D

Ell4 Test Problems

Test Problem - f2

Domain Ω: Figure D.2

$$\begin{cases} \Delta^2 u = 288x^2y^2 - 48y + 8 + 72x^2 + 24y^4 - 288x^2y \\ \qquad\qquad +72y^2 - 288xy^2 + 288xy - 48y^3 - 48x + 24x^4 - 48x^3 & \text{in } \Omega \\ u = \partial_n u = 0 & \text{on } \Gamma \end{cases}$$

Exact solution: $u = x^2y^2\left(1-x\right)^2\left(1-y\right)^2$.
This problem is a standard biharmonic test problem.

Test Problem - f3

Domain Ω: Figure D.2

$$\begin{cases} \Delta^2 u = 24\pi^4 - 40\pi^4\left(\cos(\pi x)\right)^2 - 40\pi^4\left(\cos(\pi y)\right)^2 + 64\pi^4\left(\cos(\pi x)\right)^2\left(\cos(\pi y)\right)^2 & \text{in } \Omega \\ u = \partial_n u = 0 & \text{on } \Gamma \end{cases}$$

Exact solution: $u = \left(\sin(\pi x)\right)^2\left(\sin(\pi y)\right)^2$.
This problem is a standard biharmonic test problem.

Test Problem - f4

Domain Ω: Figure D.2

$$\begin{cases} \Delta^2 u = -16\cos(2\pi x)\pi^4 + 64\cos(2\pi x)\pi^4\cos(2\pi y) - 16\cos(2\pi y)\pi^4 & \text{in } \Omega \\ u = \partial_n u = 0 & \text{on } \Gamma \end{cases}$$

Exact solution: $u = \left(1-\cos(2\pi x)\right)\left(1-\cos(2\pi y)\right)$.

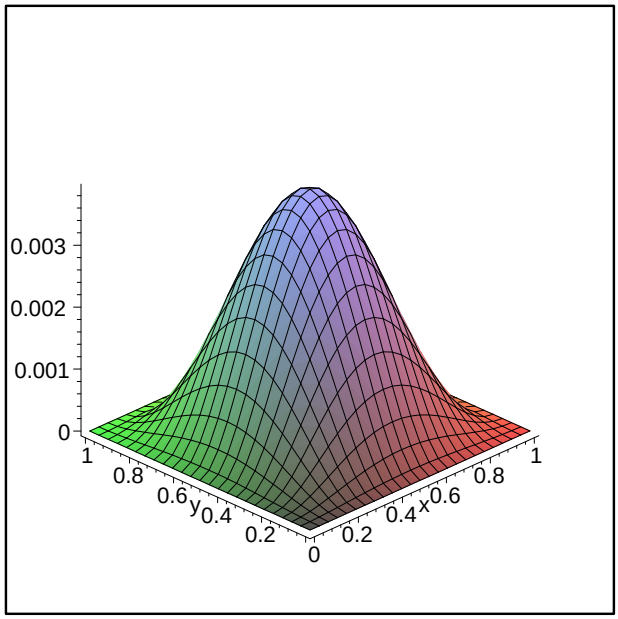


Figure D.1: f2 Exact Solution

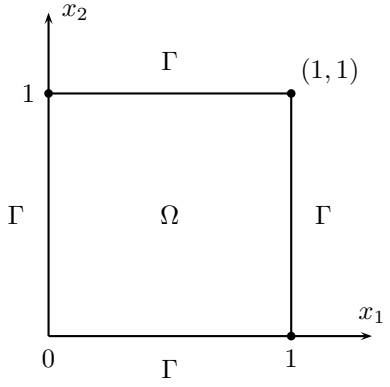


Figure D.2: Square Domain

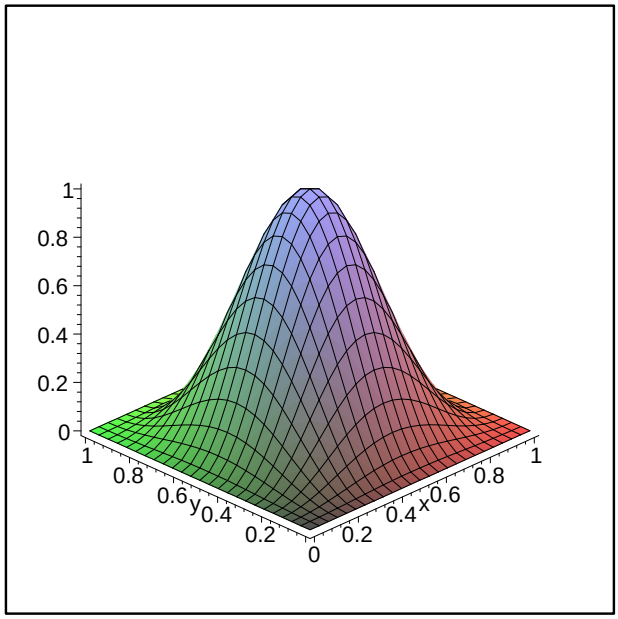


Figure D.3: f3 Exact Solution

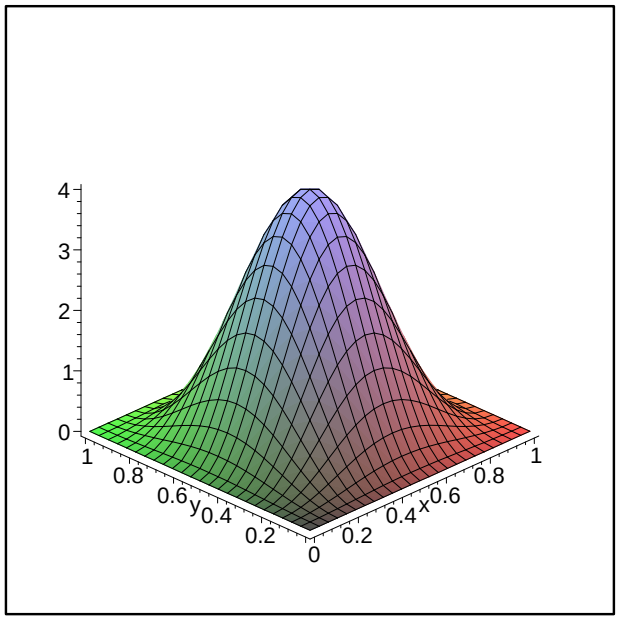


Figure D.4: f4 Exact Solution

This problem is a standard biharmonic test problem. This problem is similar to f3, differing in magnitude.

Vita

Michael Arthur Saum was born in Moline, Illinois on March 29, 1958. He graduated from Parkway West High School in Ballwin, Missouri in 1976. He attended Purdue University in West Lafayette, IN from 1976–1980, leaving the university without a degree. While at Purdue and majoring in Nuclear Engineering, he was a co-op student with Tennessee Valley Authority in Chattanooga, Tennessee from 1977–1979, working in the Methods Development Section of the Nuclear Fuels Planning Branch.

He moved to Brooklyn, New York in 1980 and obtained a job with American Electric Power in the Nuclear Engineering Division. While in New York, he attended Polytechnic Institute of New York, but was not able to finish degree requirements before moving with AEP to Columbus, Ohio in 1983. He obtained valuable work experience with AEP and while in Columbus, he finally obtained a B.A. in Mathematics from Capital University in 1993. In 1996, he accepted a severance package from AEP, leaving after 16 years as a Senior Programmer Analyst. From 1996–1997 he worked as a computer consultant primarily providing UNIX support for the Honda Corporation in central Ohio.

In 1997, he returned to the world of academia and in 1999 obtained an M.S. in Mathematics from Ohio University, co-authoring one paper while there with Dr. Todd Young. In 1999, he enrolled at the University of Tennessee, Knoxville to pursue a Ph.D. in Mathematics. From 2001–2003 he worked as a research assistant under Dr. Sergey Gavrilets in Mathematical Evolutionary theory. In the Spring of 2006, he received the *Dorothea & Edgar D. Eaves Teaching Award* recognizing excellence in graduate teaching within the Mathematics department. The doctoral degree was received August 2006 under Dr. Ohannes Karakashian. In August 2006 he accepted a postdoctoral position at the University of Tennessee working with Dr. Tim Schulze.