MACHINE LEARNING BASED REDUCTION AND QUANTIFICATION OF ERROR IN IMAGE CLASSIFICATION PIPELINES

Aritra Chowdhury

Submitted in Partial Fullfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

Examining committee:
Dr. Bulent Yener, Chair
Dr. Malik Magdon-Ismail
Dr. Charles V. Stewart
Dr. Alberto Santamaria-Pang

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York

[August 2018]
Submitted June 2018

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

ABSTRACT

Image classification is an important problem. It is an approach of pattern recognition in computer vision. The goal of the approach is to differentiate between different classes of images based on the quantification of contextual information in the images. Image classification is performed using data analytic pipelines. These pipelines are organized as interdependent and individual components. These components include image acquisition, image preprocessing, feature extraction, feature preprocessing, dimensionality reduction, learning algorithms. More steps may be added or removed from the pipeline. The quality of the image classification pipeline is measured by the error in classification of new image instances. This error is an accumulation of the errors introduced from different parts of the pipeline starting from the acquisition of the image from the sample to the algorithm used for performing the classification. This error is propagated down the pipeline which finally accumulated in the form of the aforementioned classification error. In this thesis, we attempt to reduce, quantify and understand the error in image classification pipelines.

In the first part, we try to reduce the error in a holistic manner. The problem selected for demonstrating this is that of automatic microstructure recognition. The dataset for this problem is from the domain of material science. It consists of images of microstructures in the micrometer scale. We perform two classification tasks. The first task is of differentiating between dendritic and non-dendritic microstructures. The second task is for differentiating between longitudinal and transverse cross sections of microstructures. The objective for both the classification tasks is to find the best set of algorithms and hyper-parameters for minimizing the classification error. The approach used was an exhaustive search of the configuration space of algorithms and corresponding hyperparameters for three stages of the pipeline - feature extraction, dimensionality reduction and learning algorithms. This exhaustive grid search is able to perform classification of the two tasks with a sufficiently high degree of accuracy.

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Outline

1.2 Contributions

CHAPTER 2
A MACHINE LEARNING APPROACH TO QUANTIFYING NOISE IN MEDICAL IMAGES

2.1 Introduction

As advances in medical imaging technology are resulting in significant growth of biomedical image data, new techniques are needed to automate the process of identifying images of low quality. Automation is needed because it is very time consuming for a domain expert such as a medical practitioner or a biologist to manually separate good images from bad ones. While there are plenty of de-noising algorithms in the literature, their focus is on designing filters which are necessary but not sufficient for determining how useful an image is to a domain expert. Thus a computational tool is needed to assign a score to each image based on its perceived quality. In this paper, we introduce a machine learning-based score and call it the Quality of Image (QoI) score. The QoI score is computed by combining the confidence values of two popular classification techniques?support vector machines (SVMs) and Nave Bayes classifiers. We test our technique on clinical image data obtained from cancerous tissue samples. We used 747 tissue samples that are stained by four different markers (abbreviated as CK15, pck26, E_cad and Vimentin) leading to a total of 2,988 images. The results show that images can be classified as good (high QoI), bad (low QoI) or ugly (intermediate QoI) based on their QoI scores. Our automated labeling is in agreement with the domain experts with a bi-modal classification accuracy of 94 %, on average. Furthermore, ugly images can be recovered and forwarded for further post-processing.

2.2 A Review of Adjoint-Based Error Representations

In this section, a brief review of the derivation of adjoint-based error representations is provided for Galerkin finite element methods as outlined by Becker and Rannacher [7], and for stabilized finite element methods as outlined by Cyr et al. [8]. This review is inteded to give context and serve as a road map for the remaining sections in this chapter.

This chapter previously appeared as: A. Chowdhury, K. S. Aggour, S. M. Gustafson, B. Yener, "A Machine Learning Approach to Quantifying Noise in Medical Images." *Medical Imaging 2016: Digital Pathology*, vol. 9791, pp. 979110U, 2016.

Let $\mathcal{S}$ and $\mathcal{V}$ be Hilbert spaces, $\mathcal{R}_g : \mathcal{S} \to \mathcal{V}$ and $\mathcal{R}_\tau : \mathcal{S} \to \mathcal{V}$ be semilinear forms that are linear in their first argument and potentially nonlinear in their second argument. Let $\mathcal{S}^H \subset \mathcal{S}$ and $\mathcal{V}^H \subset \mathcal{V}$ be classical finite element function spaces, where H is a mesh-dependent parameter that denotes the fineness of the discretization. We introduce the following variational abstract model problem: find $u \in \mathcal{S}$ such that

$$\mathcal{R}_g(w; u) = 0 \quad \forall w \in \mathcal{V}. \tag{2.1}$$

Similarly, we introduce the following abstract *adjoint problem*: find $z \in \mathcal{V}$ such that

$$\mathcal{R}'_g[u^H](w, z) = J'[u^H](w) \quad \forall w \in \mathcal{V}, \tag{2.2}$$

where the prime indicates Fréchet linearization about the argument in square brackets, which can equivalently be expressed as the Gâteaux derivatives

$$J'[u^H](w) := \frac{d}{d\epsilon} J(u^H + \epsilon w)\big|_{\epsilon=0}, \tag{2.3}$$

and

$$\mathcal{R}'_g[u^H](w, z) := \frac{d}{d\epsilon} \mathcal{R}_g(z; u^H + \epsilon w)\big|_{\epsilon=0}. \tag{2.4}$$

Here, $u^H \in \mathcal{S}^H$ denotes some finite element approximation to the true solution u. The purpose of the adjoint problem is to relate the original problem of interest to the functional quantity J, and it is this relationship that allows us to derive adjoint-based error representations. Further, we note that the adjoint solution z can be interpreted as the sensitivity of the QoI to perturbations in the primal PDE residual [9].

2.2.1 Galerkin Finite Element Methods

The corresponding Galerkin finite element formulation of the abstract problem (2.1) can be stated as: find $u^H \in \mathcal{S}^H$ such that

$$\mathcal{R}_g(w^H; u^H) = 0 \quad \forall w^H \in \mathcal{V}^H. \tag{2.5}$$

Let $e := u - u^H$ denote the discretization error. We can then derive an error representation for the functional J in terms of the adjoint solution z as follows:

$$\begin{aligned} J(u) - J(u^H) &= J'[u^H](e) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g'[u^H](e, z) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g(z; u) - \mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z - z^H; u^H) + \mathcal{O}(e^2). \end{aligned} \tag{2.6}$$

Here, the first equality is due to the linearization [7] of the functional J, the second equality is due to the introduced adjoint problem (2.2), the third equality is due to the linearization [7] of the residual $\mathcal{R}_g$, the fourth equality holds due to the definition of the abstract primal problem (2.1), and the fifth equality is due to Galerkin orthogonality, where z^H denotes the interpolant of z onto the space $\mathcal{V}^H$. In reference to the notation introduced, the total residual semilinear form $\mathcal{R}$ is given as $\mathcal{R} = \mathcal{R}_g$.

2.2.2 Stabilized Finite Element Methods

A corresponding stabilized finite element method of the abstract problem (2.1) can be expressed as: find $u^H \in \mathcal{S}^H$ such that

$$\mathcal{R}_g(w^H; u^H) + \mathcal{R}_\tau(w^H; u^H) = 0 \quad \forall w^H \in \mathcal{V}^H. \tag{2.7}$$

Here $\mathcal{R}_\tau$ denotes a consistent *stabilization residual* that adds stability to the numerical scheme. We say that the stabilization is *consistent* if $\mathcal{R}_\tau(w^H; u) \to 0$ as $H \to 0$.

Again, we let $e := u - u^H$ denote the discretization error, and derive an error representation for the functional J as follows

$$\begin{aligned} J(u) - J(u^H) &= J'[u^H](e) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g'[u^H](e, z) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g(z; u) - \mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z - z^H; u^H) + \mathcal{R}_\tau(z^H; u^H) + \mathcal{O}(e^2). \end{aligned} \tag{2.8}$$

Here, the first four equalities are obtained exactly as in the corresponding Galerkin finite element method. However, when we subtract the interpolant z^H of the adjoint solution z in the fifth equality, an additional term remains because the numerical scheme (2.7) lacks Galerkin orthogonality. In reference to the notation introduced, the total semilinear form $\mathcal{R}$ is given as $\mathcal{R} = \mathcal{R}_g + \mathcal{R}_\tau$.

2.3 Software Components

An adaptive adjoint-based simulation requires the implementation and coordination of a number of non trivial components. Namely, the solution of a primal problem, the construction and solution of an auxiliary adjoint problem, an enrichment of the adjoint solution, the estimation and localization of the error, and mesh adaptation are all required steps in the adjoint-based adaptive process.

To implement each of these components for effective execution on parallel machines, we make use of two state of the art software suites. The first is PUMI [10], which contains tools to support unstructured mesh services on massively parallel machines. In particular, PUMI provides all of the necessary machinery to store, query, adapt, and dynamically load balance parallel unstructured meshes via a collection of modern `C` and `C++` libraries. The second is Trilinos [11], [12], which provides a large variety of `C++` packages to support multiphysics simulations on parallel machines. In particular, Trilinos provides the ability to store and solve sparse parallel linear systems, as well as tools to perform automatic differentiation.

Using these two software suites as building blocks, we have written a new `C++` application for adjoint-based error estimation and mesh adaptation with an emphasis on nonlinear solid mechanics. We have called this application Goal [4]. Below, we describe how these software components are utilized for each portion of the adaptive adjoint-based process, where the analysis is automated based only on the inputs of a semilinear form $\mathcal{R}$ and a functional QoI J.

2.3.1 The Primal Problem

Based on an implemented weighted residual operator $\mathcal{R}$, the Goal application computes element-level residual vectors and element-level Jacobian matrices. The element-level Jacobian matrices are computed via automatic differentiation using the Trilinos library Sacado. Sacado provides efficient automatic differentiation using a `C++` meta-programming technique

called expression templates [13].

After the computation of a single element's residual vector and Jacobian matrix, the Goal application performs a finite element assembly step to sum contributions to the global residual vector and global Jacobian matrix. To store and modify the global linear algebra objects, we utilize the Tpetra library provided by Trilinos. In particular, the Jacobian matrix is stored as a sparse compressed row storage matrix in parallel.

The primal problem is solved via Newton's method, which requires iterative evaluations of the global residual vector and Jacobian matrix. For each Newton iteration, a global linear system must be solved. We solve this linear system iteratively in parallel using either a CG or GMRES solver provided the Trilinos library Belos [14]. Additionally, we perform algebraic multigrid preconditioning using the Trilinos library MueLu [15].

Once the primal problem has been solved, we utilize the PUMI library APF to store the finite element solution information at nodes and if necessary secondary solution information at integration points. Additionally, the APF library is used to provide shape function information and to query stored solution information during residual and Jacobian evaluations. Throughout the entire solution process, the PUMI mesh data structure is utilized to query mesh specific information.

2.3.2 The Adjoint Problem

To solve the adjoint problem in a richer finite element space than the one used for the primal problem, we make use of the underlying PUMI mesh data structure [16] and the PUMI MeshAdapt software to create and store a uniformly refined nested mesh with parent-child relations back to the original mesh. This relational information is implemented in the Goal application, as it falls outside of the normal intended use case of the MeshAdapt software, which concerns itself with fully unstructured conforming mesh adaptation via edge splits, swaps, and collapses. However, the flexibility of the PUMI software allows us to additionally construct data structures similar to those used in traditional adaptive mesh refinement (AMR) with little implementation effort. Using the APF library and the parent-child relational information, we are able to interrogate stored solution fields on both the parent and nested meshes, which is required during the assembly of the adjoint problem.

On this finer mesh, element-level Jacobian matrices are computed using the Sacado library, based on the Goal implementation of the operator $\mathcal{R}$. Additionally, element-level

derivatives of the functional quantity of interest J are computed with respect to degrees of freedom of the problem, resulting in an element-level functional derivative vector.

After the computation of a single element's Jacobian matrix and functional derivative vector, the Goal application performs a finite element assembly step to sum contributions to the global discrete adjoint matrix and the global functional derivative vector. Like the primal problem, these global parallel linear objects are stored using the Tpetra library. The global discrete adjoint operator and functional derivative vector fully define the linearized adjoint problem, which we again precondition with algebraic multigrid techniques using the MueLu library and solve using either CG or GMRES iterations using the Belos library. The fine-space adjoint solution is then attached to the mesh using the APF library.

2.3.3 Error Estimation and Localization

The error estimation and localization routines are implemented entirely in the Goal application. The error is localized by an evaluation of the stabilized weighted residual operator $\mathcal{R}$, where the weight is chosen to be the adjoint solution multiplied by a partition of unity. This error localization is discussed in further detail in Section 2.7. Based on these element-level residual vectors, Goal performs finite element assembly of the global residual vector, which is stored as a Tpetra vector. This vector represents an adjoint-weighted residual error estimate at each mesh vertex for each PDE equation in the fine mesh. The error is attached to the vertices of the fine mesh using the APF library.

2.3.4 Mesh Adaptation

Once the error is stored on the vertices of the fine mesh, we interpolate it to element centers of the coarse mesh. The fine mesh data structures are then destroyed and the Goal application computes a mesh size field that seeks to equidistribute the error for an output mesh with N elements. This mesh size field is given as the input to the PUMI MeshAdapt software, which adapts the mesh with a sequence of edge splits, swaps, and collapses [17], [18] to satisfy the given mesh size field. As a final step, we utilize the PUMI library ParMA [19], [20] to perform diffusive load balancing to ensure parallel partitioning quality.

2.3.5 In-Memory Integration of Components

The coupling of the software components described above is done *in-memory* [21]. That is, there is no file-based communication of data from one analysis component to the next in the automated process. This in-memory coupling is a key ingredient for parallel analysis, where filesystem bandwidth is a critical bottleneck.

2.4 Template-Based Generic Programming

In this section, we provide a review of the concept of *template-based generic programming* for the evaluation and solution of PDEs [5], [6] and how it has been extended in the Goal application to automate the process of adjoint-based error estimation and mesh adaptation. For PDE applications, TBGP is broken into three major components, a *seed* or *gather* phase, a *compute* phase, and a *scatter* phase. The seed and scatter operations must be programmed specifically for each evaluation purpose. In contrast, the compute phase, where the PDE and QoI expressions are implemented, are written in a totally generic manner. Figure 2.1 pictorially represents this design philosophy.

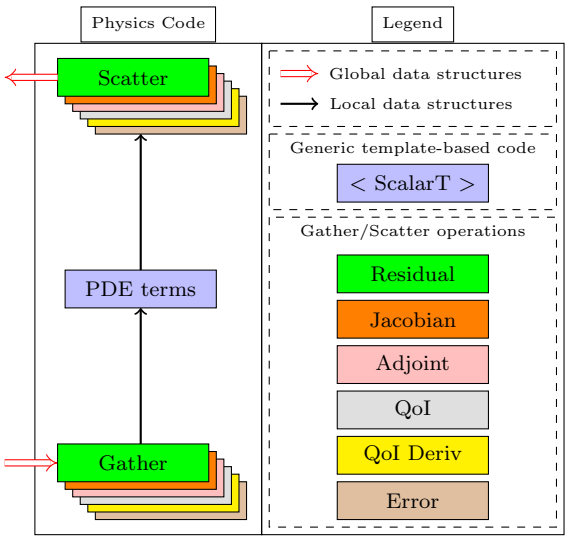

Fig. 2.1. A schematic for the generic programming model of PDEs.

We invoke this approach at the element-level, meaning for each element we perform the process: Gather → Compute → Scatter. By doing so, we reduce memory overhead and eliminate complications introduced by parallel computation [6]. Underlying the TBGP approach is the use of forward automatic differentiation (FAD) [22], which is discussed in

further detail in Appendix A. The Goal application utilizes the Trilinos library Sacado [13] to perform automatic differentiation.

Table 2.1. A list of TBGP evaluation operations used in the Goal application. In this table $\boldsymbol{u}^H$ is the primal solution vector, $\boldsymbol{u}_H^h$ is the prolongation of the solution vector to a richer space, $\boldsymbol{s}^H$ is a (potentially empty) vector of history-dependent mechanics state variables, $\boldsymbol{s}_H^h$ is the prolongation of the state to a richer space, $\boldsymbol{z}^h$ is the adjoint solution vector, $\boldsymbol{R}^H$ is the residual vector evaluated on the coarse space, $\boldsymbol{R}^h$ is the residual vector evaluated on the fine space, and J^H is the scalar QoI.

Evaluation Type	Scalar Type	Input	Output
Residual	`double`	$\boldsymbol{u}^H, \boldsymbol{s}^H$	$\boldsymbol{R}^H$
Jacobian	`Sacado::FAD`	$\boldsymbol{u}^H, \boldsymbol{s}^H$	$\frac{\partial \boldsymbol{R}^H}{\partial \boldsymbol{u}^H}$
Adjoint	`Sacado::FAD`	$\boldsymbol{u}_H^h, \boldsymbol{s}_H^h$	$\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\Big\vert_{\boldsymbol{u}_H^h}\right]^T$
QoI	`double`	$\boldsymbol{u}^H, \boldsymbol{s}^H$	J^H
QoI Deriv	`Sacado::FAD`	$\boldsymbol{u}_H^h, \boldsymbol{s}_H^h$	$\left[\frac{\partial J^h}{\partial \boldsymbol{u}^h}\Big\vert_{\boldsymbol{u}_H^h}\right]^T$
Error	`double`	$\boldsymbol{u}_H^h, \boldsymbol{s}_H^h, \boldsymbol{z}^h$	$\boldsymbol{R}^h$

The purpose of the gather/seed operation is to collect information from global storage containers and 'gather' it to local element-level data structures. Further, any FAD derivative information is *seeded* during this operation, if necessary. For each evaluation type, the gather/seed operation initializes an array that physically represent the degrees of freedom associated with the current element, and initializes FAD variables' derivative arrays to physically represent derivatives with respect to the degrees of freedom associated with the current element, when necessary. This degree of freedom array is templated on a scalar type `ScalarT`. For the Residual, QoI, and Error evaluation types, this scalar type corresponds to a `C++` double. For the remaining evaluation types, this scalar type corresponds to a `Sacado::FAD` forward automatic differentiation variable type.

The compute phase computes local contributions to the equations or expressions of interest at the element level in terms of the degrees of freedom, as collected by the gather operation. The code for the compute phase is written in an entirely generic fashion, and is templated on a scalar type `ScalarT`. Templating the code used for the compute phase, along with appropriately chosen gather and scatter operations, allows the same code to be re-used

Listing 2.1. The abstract Goal integrator class interface.

```
1  class Integrator {
2    public:
3      Integrator();
4      virtual ~Integrator();
5      std::string const& get_name() { return name; }
6      virtual void set_time(double, double) {}
7      virtual void pre_process(SolInfo*) {}
8      virtual void set_elem_set(int) {}
9      virtual void gather(apf::MeshElement*) {}
10     virtual void in_elem(apf::MeshElement*) {}
11     virtual void at_point(apf::Vector3 const&, double, double) {}
12     virtual void out_elem() {}
13     virtual void scatter(SolInfo*) {}
14     virtual void post_process(SolInfo*) {}
15   protected:
16     std::string name;
```

for the distinct evaluation purposes listed in Table 2.1.

The scatter phase takes the local element-level data evaluated in the compute phase and 'scatters' it to the appropriate global data structure, as determined by the current evaluation operation. For instance, for the residual evaluation operation, local element-level residuals are evaluated in the compute phase and then summed into appropriate locations in the global residual vector during the scatter phase. Similarly, for the Jacobian evaluation operations, local element-level Jacobians are evaluated in the compute phase and the scatter opeation sums these local contributions to appropriate locations in the global Jacobian matrix.

In the Goal application, we have considered six specific gather/scatter evaluation operations corresponding to the evaluation of the global residual vector, evaluation of the global Jacobian matrix, evaluation of the adjoint of the global Jacobian matrix, evaluation of the functional QoI, evaluation of the derivative of the functional QoI, and evaluation of localized adjoint-weighted residual error estimates. Table 2.1 lists the inputs and outputs for these specific evaluation operations.

To realize these specific gather/scatter operations, we have implemented an abstract degree of freedom class and an abstract quantity of interest class that are both templated on a scalar type `ScalarT`. This scalar type is explicitly instantiated to either be a `C++` double or a `Sacado::FAD` forward automatic differentiation variable type. Both the degree of freedom

and QoI classes are equipped with `gather` and `scatter` methods, whose behavior changes based on an input parameter given to the class constructor. For the degree of freedom class, this parameter selects gather/scatter operations for either the residual, Jacobian, adjoint Jacobian, or adjoint-weighted residual error evaluations. Similarly, for the QoI class, this input parameter selects gather/scatter operations for either the evaluation of the QoI or the derivative of the QoI with respect to the problem degrees of freedom.

Previously, TBGP has been utilized in the multiphysics code Albany [23], [24] with the capability to perform the Residual, Jacobian, Adjoint, QoI, and QoI derivative evaluation operations shown in Table 2.1. To extend the abilities of TBGP to include adjoint-based error estimation, the Goal application implements the ability to perform the Adjoint and QoI derivative evaluations in a richer finite element space, as discussed in Section 2.6, a feature not previously available in existing TBGP codes. Further, the Goal application implements a novel evaluation type for the localization of the error, referred to as the Error evaluation type in Table 2.1. For this purpose, we have implemented an abstract weighting function class whose behavior changes based on the chosen evaluation type. This class evaluates the appropriate finite element weighting function values and gradients based on linear Lagrange basis functions for the Residual, Jacobian and Adjoint evaluation types. However, for the Error evaluation type, the behavior of the weighting function class is modified such that it evaluates the value and gradient of the adjoint solution z^h multiplied by a partition of unity. This abstraction of the weighting function class allows us to re-use the PDE implementation of the semilinear form $\mathcal{R}$ to assemble a residual vector $\boldsymbol{R}^h$ that represents an adjoint-weighted residual error estimate at each mesh vertex for each PDE equation in the richer finite element space, which is then used to drive mesh adaptation.

Listing 2.1 demonstrates the abstract integrator interface that has been implemented in the Goal application. The abstract degree of freedom, QoI, and weighting function classes inherit from this base class. For each of these classes, the `gather` and `scatter` methods are implemented specifically for each appropriate evaluation type. The PDE equations in the Goal application are written as a combination of `Goal::Integrator`s. Given an ordered array of integrators, the Goal application performs finite element assembly for every evaluation type in a generic manner, as outlined by Algorithm 1.

Algorithm 1 Assembly algorithm used in the Goal application

Given a mesh M and an ordered array of integrators I:
Call `pre_process` for each integrator i in I.
for each element set es in mesh M **do**
  Call `set_elem_set` for each integrator i in I.
  for each element e in element set es **do**
    Call `gather` for each integrator i in I.
    Call `in_elem` for each integrator i in I.
    for each integration point ip in element e **do**
      Call `at_point` for each integrator i in I.
    end for
    Call `out_elem` for each integrator i in I.
    Call `scatter` for each integrator i in I.
  end for
end for
Call `post_process` for each integrator i in I.

Listing 2.2. Poisson residual.

```
template <typename ScalarT>
void Residual<ScalarT>::at_point(
    apf::Vector3 const& p, double ipw, double dv) {
  apf::Vector3 x(0,0,0);
  apf::mapLocalToGlobal(elem, p, x);
  double fval = eval(f, x[0], x[1], x[2], 0.0);
  for (int n = 0; n < u->get_num_nodes(); ++n)
  for (int i = 0; i < num_dims; ++i)
    u->resid(n) += u->grad(i) * w->grad(n, i) * ipw * dv;
  for (int n = 0; n < u->get_num_nodes(); ++n)
    u->resid(n) -= fval * w->val(n) * ipw * dv;
}
```

2.5 The Primal Problem

2.5.1 Galerkin Finite Element Methods

We recall the definition of the abstract Galerkin finite element model problem, given by equation (2.5). In this context, the weighted residual form $\mathcal{R}_g$ is implemented in the Goal application. As an example, Listing 2.2 demonstrates the implementation of the Poisson residual $\mathcal{R}_g(w;u) := (\nabla w, \nabla u) - (w, f)$ in the Goal application.

Listing 2.3. Pressure stabilization residual for mechanics.

```
1  template <typename ScalarT>
2  void Stabilization<ScalarT>::at_point(
3      apf::Vector3 const&, double ipw, double dv) {
4    double h = get_size(mesh, elem);
5    double tau = 0.5*c0*h*h/mu;
6    auto J = k->get_det_def_grad();
7    auto F = k->get_def_grad();
8    auto Cinv = inverse(transpose(F)*F);
9    for (int n = 0; n < p->get_num_nodes(); ++n)
10   for (int i = 0; i < num_dims; ++i)
11   for (int j = 0; j < num_dims; ++j)
12     p->resid(n) += tau * J * Cinv(i, j) *
13       p->grad(i) * w->grad(n, j) * ipw * dv;
14 }
```

2.5.2 Stabilized Finite Element Methods

We recall the definition of the abstract stabilized finite element model problem, given by equation (2.7). In this context, both the weighted residual statement $\mathcal{R}_g$ and the stabilized weighted residual form $\mathcal{R}_\tau$ are implemented in the Goal application. As an example, Listing 2.3 demonstrates the implementation of the pressure stabilization [25] residual $\mathcal{R}_\tau(w;u)$ term used in the Goal application for finite deformation solid mechanics. This stabilization term is discussed in greater detail in Section 2.10.2.

2.5.3 Automated Solution Based on Residual Implementation

For each element, we compute element level Jacobian matrices by applying automatic differentiation [22] to element-level contributions to the residual vector. For example, Listing 2.2 demonstrates how contributions to the element-level Poisson's equation residual $\mathcal{R}(w;u) = (\nabla w, \nabla u) - (w, f)$ are implemented. The element level Jacobian matrices are then assembled into the global system Jacobian operator $\boldsymbol{\mathcal{J}}^H \in \mathbb{R}^{N\times N}$, given by

$$\boldsymbol{\mathcal{J}}^H = \frac{\partial \boldsymbol{R}^H(\boldsymbol{u}^H)}{\partial \boldsymbol{u}^H} \tag{2.9}$$

Listings 2.2 and 2.3 both demonstrate how element-level contributions to the semilinear forms $\mathcal{R}_g$ and $\mathcal{R}_\tau$, respectively, are computed in the Goal application. Notice that this code is templated on a scalar type `ScalarT`. When the scalar type is chosen as a `C++ double`,

element-level contributions to the residual vector $\boldsymbol{R}^H$ are computed. When the scalar type is chosen as a Sacado forward automatic differentiation variable, element-level contributions to the Jacobian matrix $\boldsymbol{\mathcal{J}}^H$ are computed. This illustrates a key concept of template-based generic programming, in that the governing equations need only be implemented once to compute a variety of additional information.

With the ability to fully assemble the Jacobian matrix $\boldsymbol{\mathcal{J}}^H$ and the residual vector $\boldsymbol{R}^H$, we solve the governing equations with Newton's method, where we iterate over the steps

$$\begin{aligned} \boldsymbol{\mathcal{J}}^H(\boldsymbol{u}_k^H)\,\delta\boldsymbol{u}_k^H &= -\boldsymbol{R}^H(\boldsymbol{u}_k^H) \\ \boldsymbol{u}_{k+1}^H &= \boldsymbol{u}_k^H + \delta\boldsymbol{u}_k^H, \end{aligned} \tag{2.10}$$

unitl the convergence criterion $\|\boldsymbol{R}^H(\boldsymbol{u}^H)\|_2 < \epsilon$ is met for a user-specified tolerance ϵ. Here $\boldsymbol{u}_k^H$ denotes the solution vector at the k^{th} iteration obtained by solving the Newton linear system. For linear variational problems, we simply restrict ourselves to a single Newton linear solve, which reduces exactly to classical FEM assembly for linear problems.

2.6 The Adjoint Problem

2.6.1 A Richer Space via Uniform Refinement

The adjoint solution must be represented in a richer space than the one used for the primal problem to obtain meaningful error estimates. There are several strategies that are commonly used to obtain such a representation. First, the adjoint problem can be solved in the same finite element space as the primal problem and then be enriched to a higher order polynomial space [7] or a nested mesh [26] by some local patch-wise operation, or variational multiscale enrichment [27] can be used in the context of stabilized finite elements. Alternatively, the adjoint problem can be solved in a higher order polynomial space [28], which we will refer to as p-enrichment. As a final option, the adjoint problem can be solved on a uniformly refined mesh [29], which we will refer to as h-enrichment.

In this work, we choose the h-enrichment approach for several reasons. First, we would like the adjoint solution to be as accurate as possible for error estimation purposes, so we choose to solve the adjoint problem in a globally richer finite element space. Additionally, for stabilized finite element methods, the use of p-enrichment would in general necessitate the use of higher order stabilization terms that vanish for lower-order finite element methods with

simplical elements. These higher order terms are typically more difficult to implement than their lower order counterparts. Further, we remark that higher-order stabilized finite element methods are rarely used in practice, as stable higher-order mixed methods can usually be derived with fewer overall degrees of freedom [30]. Finally, we note that the unstructured mesh adaptation capabilities of the PUMI software make the h-enrichment approach readily available.

In the present context, we consider the term *uniform refinement* for triangles and tetrahedra to mean splitting each edge of the parent element at its midpoint. Or, in other words, creating new edges by by connecting the midpoints of the parent element's existing edges. The uniform refinement of a triangle results in 4 nested triangles and the uniform refinement of a tetrahedron results in 8 nested tetrahedra.

We have denoted the trial and test spaces used for the primal problem as $\mathcal{S}^H$ and $\mathcal{V}^H$, respectively. We denote the trial and test spaces on the uniformly nested mesh as $\mathcal{S}^h$ and $\mathcal{V}^h$, respectively, where $h < H$ is representative of a finer mesh size. Figure 2.2 illustrates the discretization for the coarse and fine trial and test spaces defined for a three dimensional geometry with a complex void inclusion.

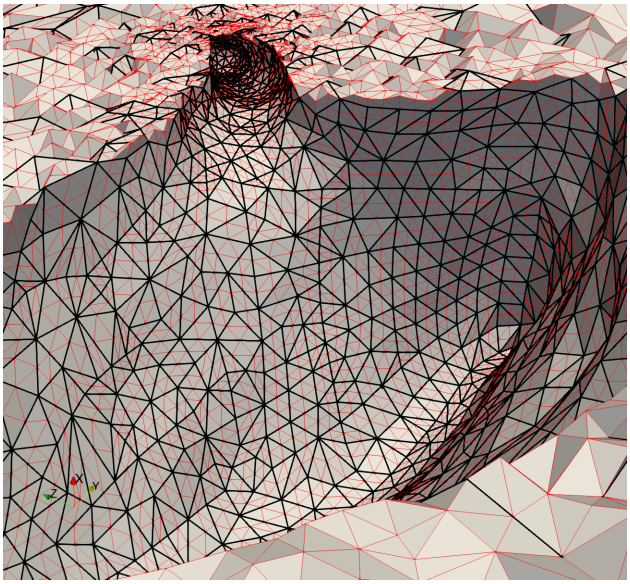

Fig. 2.2. Example of a nested mesh (red edges) obtained via a uniform refinement of a base mesh (black edges) in three dimensions.

2.6.2 Discrete Adjoint Approximation

Let $\boldsymbol{R}^h : \mathbb{R}^n \to \mathbb{R}^n$ denote the residual form of the system of nonlinear algebraic equations arising either from the Galerkin (2.5) or stabilized (2.7) model problem posed on the uniformly nested mesh. Let $\boldsymbol{u}_H^h := I_H^h \boldsymbol{u}^H$ denote the prolongation of the primal finite element solution onto the richer space $\mathcal{S}^h$ via interpolation, Let $J^h : \mathbb{R}^n \to \mathbb{R}$ denote the discretization of the functional QoI on the uniformly nested fine space. We approximate the adjoint problem (2.2) in a discrete manner [9], [31]–[33], by solving

$$\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\bigg|_{u_H^h}\right]^T \boldsymbol{z}^h = \left[\frac{\partial J^h}{\partial \boldsymbol{u}^h}\bigg|_{u_H^h}\right]^T . \tag{2.11}$$

This allows us to automate the process of solving the adjoint problem, as discussed below. Here $\boldsymbol{z}^h \in \mathbb{R}^n$ denotes the adjoint solution vector on the nested discretization.

2.6.3 Automated Solution Based on Residual Formulation

The construction of the Jacobian transpose matrix $\left[\partial \boldsymbol{R}^h / \partial \boldsymbol{u}^h\right]^T$ is performed in the same automated manner as the Jacobian for the primal problem. That is, for each element, we compute consistent element tangent stiffness matrices via automatic differentiation of element-level contributions to the residual vector. However, during the *scatter* phase of the template-based generic programming process, we transpose the element-level tangent matrices and sum them into global Jacobian adjoint matrix. The computation of the Jacobian adjoint is done using the same templated code that is used to compute the primal residual vector and the Jacobian matrix, as illustrated by listings 2.2 and 2.3.

Similarly, the construction of the functional derivative vector $\left[\partial J^h / \partial \boldsymbol{u}^h\right]^T$ is done by evaluating derivatives of element-level contributions to the functional via automatic differentiation. This results in element-level derivative vectors that are then assembled into the global functional derivative vector. Listings 2.4 and 2.5 illustrate the implementation of two quantities of interest in the Goal application. Once the Jacobian transpose matrix and functional derivative vector have been assembled, we solve the adjoint problem (2.11) using a sparse iterative solver in parallel.

2.7 Error Estimation

2.7.1 Two-Level Error Estimates

Following Venditti and Darmofal [31]–[33], we review adjoint-based error estimation using two discretization levels. The discrete residual form of the governing equations for a Galerkin (2.5) finite element method or a stabilized finite element method (2.7) posed on the fine space can be expressed as

$$\boldsymbol{R}^h(\boldsymbol{u}^h) = \boldsymbol{0}. \tag{2.12}$$

Taking Taylor expansions of the discrete residual $\boldsymbol{R}^h$ evaluated on the fine space and the discrete functional J^h evaluated on the fine space about the point $\boldsymbol{u}_H^h$ yields

$$\boldsymbol{R}^h(\boldsymbol{u}^h) = \boldsymbol{R}^h(\boldsymbol{u}_H^h) + \left[\left.\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\right|_{\boldsymbol{u}_H^h}\right](\boldsymbol{u}^h - \boldsymbol{u}_H^h) + \dots \tag{2.13}$$

and

$$J^h(\boldsymbol{u}^h) = J^h(\boldsymbol{u}_H^h) + \left[\left.\frac{\partial J^h}{\partial \boldsymbol{u}^h}\right|_{\boldsymbol{u}_H^h}\right](\boldsymbol{u}^h - \boldsymbol{u}_H^h) + \dots \tag{2.14}$$

respectively.

Using equation (2.12), the discretization error between the two spaces can be approximated to first order as

$$(\boldsymbol{u}^h - \boldsymbol{u}_H^h) \approx -\left[\left.\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\right|_{\boldsymbol{u}_H^h}\right]^{-1} \boldsymbol{R}^h(\boldsymbol{u}_H^h), \tag{2.15}$$

which can then be substituted into the functional Taylor expansion (2.14) to obtain the so-called *adjoint weighted residual*

$$J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h) \approx -\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h) \tag{2.16}$$

where $\boldsymbol{z}^h$ is the solution to the adjoint problem (2.11).

2.7.2 Modified Functional Error Estimate

Assume that the QoI converges at the rate k, such that $J - J^h(\boldsymbol{u}_H^h) = cH^k$ and $J - J^h(\boldsymbol{u}^h) = ch^k$, where J denotes the exact value of the QoI. If the fine space is obtained via uniform mesh refinement, then the ratio of the fine mesh size to the coarse mesh size is given as $\frac{h}{H} = \frac{1}{2}$. Consider the ratio

$$\begin{aligned}\frac{J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h)}{J - J^h(\boldsymbol{u}_H^h)} &= \frac{\left[J - J^h(\boldsymbol{u}_H^h)\right] - \left[J - J^h(\boldsymbol{u}^h)\right]}{J - J^h(\boldsymbol{u}_H^h)} \\ &= \frac{cH^k - ch^k}{cH^k} \\ &= 1 - \left(\frac{h}{H}\right)^k \\ &= 1 - \left(\frac{1}{2}\right)^k\end{aligned} \tag{2.17}$$

in the limit as $H \to 0$ [9]. We call this ratio $\alpha := 1 - (1/2)^k$. Let η denote an approximation to the functional error $J - J^h(\boldsymbol{u}_H^h)$. Let $\mathcal{I}$ denote the effectivity index given by

$$\mathcal{I} = \frac{\eta}{J - J^h(\boldsymbol{u}_H^h)}. \tag{2.18}$$

We would like to obtain error estimates η that lead to effectivity indices of $\mathcal{I} = 1$ as $H \to 0$. To this end, we recall $J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h) \approx -\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h)$ from equation (2.16) to obtain the scaled adjoint weighted residual error estimate

$$\eta = -\frac{1}{\alpha}\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h). \tag{2.19}$$

2.7.3 Error Localization for Galerkin Methods

Following the approach of Richter and Wick [3], we introduce a partition of unity ϕ_i, such that $\sum_i \phi_i = 1$, into the weighting function slot for the error estimate to localize the error. In this work, the partition of unity is realized as linear Lagrange basis functions. This yields local error contributions η_i at the n_{vtx} mesh vertices in the mesh. Let $z^h \in \mathcal{V}^h$ be the finite element solution obtained by solving the discrete adjoint problem (2.11). We assume that this solution well approximates the continuous adjoint problem (2.2), such that $z \approx z^h$. Let z^H denote the interpolant of z^h onto the coarse space $\mathcal{V}^H$. Recalling the error representation (2.6) for Galerkin finite elements, we obtain partition of unity-based

correction indicators η_i in the following manner

$$J(u) - J(u^H) \approx \sum_{i=1}^{n_{vtx}} \underbrace{-\mathcal{R}_g((z^h - z^H)\phi_i \; ; \; u^H)}_{\eta_i}. \tag{2.20}$$

2.7.4 Error Localization for Stabilized Methods

Error localization for the stabilized finite element formulation (2.7) proceeds in the same manner as the previous section. Let $z^h \in \mathcal{V}^h$ denote the finite element solution obtained by solving the discrete adjoint problem (2.11) and let z^H denote the interpolant of z^h onto the coarse space $\mathcal{V}^H$. Introducing a partition of unity into the error representation (2.8) for stabilized finite element methods with the approximation $z \approx z^h$ yields the vertex-based correction indicators η_i:

$$J(u) - J(u^H) \approx \sum_{i=1}^{n_{vtx}} \underbrace{-\mathcal{R}_g((z^h - z^H)\phi_i \; ; \; u^H) + \mathcal{R}_\tau(z^H\phi_i \; ; \; u^H)}_{\eta_i}. \tag{2.21}$$

Once correction indicators η_i have been evaluated, an approximate upper bound $\hat{\eta}$ for the error is computed by summing the absolute value of the error contributions over all mesh vetrices

$$\hat{\eta} = \sum_{i=1}^{n_{vtx}} |\eta_i|. \tag{2.22}$$

2.7.5 Automated Error Localization Based on Residual Implementation

During the assembly of the adjoint problem (2.11), the evaluation of element-level contributions to the residual vector evaluated on the fine space $\boldsymbol{R}^h(\boldsymbol{u}_H^h)$ are necessarily computed by the machinery of forward automatic differentiation. Thus, during the `scatter` phase for the adjoint problem computation, we additionally sum element-level contributions to the fine residual to assemble the global vector $\boldsymbol{R}^h(\boldsymbol{u}_H^h)$. This, along with the solution $\boldsymbol{z}^h$ to the adjoint problem (2.11) provides enough information to compute the adjoint-weighted residual estimate (2.19) in an automated fashion.

Again, we let $z^h \in \mathcal{V}^h$ denote the finite element solution to the discrete adjoint problem (2.11) and let z^H denote the interpolant of z^h onto the coarse space $\mathcal{V}^H$. We refer again to Listings 2.2 and 2.3, which illustrate implementations of Galerkin and stabilized semilinear

forms $\mathcal{R}_g$ and $\mathcal{R}_\tau$, respectively, in the Goal application. Specifically, we remark that these residual evaluations contain the evaluation of weighting functions and their derivatives, given with calls to the methods `w->val(node)` and `w->grad(node, dim)`, respectively. To localize the error in an automated fashion, we override the calls to these methods such that they return values of the adjoint solution multiplied by a partition of unity. For instance, at a given reference location $\boldsymbol{\xi}$ in a given element, the partition of unity-based weight for the Galerin residual $\mathcal{R}_g$ is computed as

$$\texttt{w->val(n)} = \left[(z^h - z^H) \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}, \tag{2.23}$$

and its corresponding gradient is computed as

$$\texttt{w->grad(n)} = \nabla \left[(z^h - z^H) \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}. \tag{2.24}$$

Similarly, for the stabilized residual $\mathcal{R}_\tau$, the partition of unity-based adjoint weight is computed as

$$\texttt{w->val(n)} = \left[z^H \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}, \tag{2.25}$$

and its corresponding gradient is computed as

$$\texttt{w->grad(n)} = \nabla \left[z^H \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}. \tag{2.26}$$

In this manner, we have introduced partition of unity-based adjoint weights that have the same data type as the weights used for the computation of the primal and adjoint problems.

Using the adjoint weights in the error localization evaluation results in element-level residual vectors that correspond to contributions to the localized correction indicators η_i. During the `scatter` phase of the error localization evaluation, we sum these element level contributions to the appropriate mesh vertices to compute the localized correction indicators η_i.

2.8 Mesh Adaptation

Given localized correction indicators η_i at mesh vertices, we compute element-level correction indicators η_e for $e = 1, 2, \ldots, n_{el}$, where n_{el} is the number of elements in the coarse discretization, by interpolating the vertex-based indicators to element centers and taking the result's absolute value.

We then specify a *mesh size field* that defines the desired value of edge lengths over the mesh. From a high-level, we would like to specify this size field such that areas of the mesh that contribute strongly to the error in the QoI are refined, and areas of the mesh that are insensitive to the error are coarsened. Following Boussetta et al. [34], we specify a size field that attempts to equidistribute the error in an output adapted mesh with N target elements. Let p be the polynomial interpolant order for the chosen finite element method. In the present setting, $p = 1$. We first define the global quantity G as

$$G = \sum_{e=1}^{n_{el}} (\eta_e)^{\frac{2d}{2p+d}}. \tag{2.27}$$

This global quantity arises by considering *a priori* convergence rates for the input mesh and attempting to find an optimal mesh size for an output mesh with N elements. [34]. With this global quanity, new element sizes H_e^{new} are computed by scaling the previous element size H_e

$$H_e^{\text{new}} = \left(\frac{G}{N}\right)^{\frac{1}{d}} (\eta_e)^{\frac{-2}{2p+d}} H_e. \tag{2.28}$$

Finally, to prevent excessive refinement or coarsening in a single adaptive step, we clamp the element size such that it is no smaller than one quarter and no greater than twice the previous element size. This clamping is performed to ensure that mesh adaptation is being driven by accurate error indicators.

$$\frac{1}{4} \leq \frac{H_e^{\text{new}}}{H_e} \leq 2. \tag{2.29}$$

2.9 Quantities of Interest

In this section, we review three quantities of interest that we have implemented in the Goal application. One benefit of the current automated approach is that additional

quantities of interest can be rapidly prototyped and investigated with relative ease. Here, we refer to the domain discretized by the finite element mesh as Ω.

2.9.1 Point-Wise Solution Component

First, we consider the evaluation of a component u_i of the solution u at a given spatial localtion $\boldsymbol{x}$. This functional can be expressed as

$$J(u) = \int_{\Omega} \delta(\boldsymbol{x} - \boldsymbol{x}_0)\, u_i \, \mathrm{d}\Omega, \tag{2.30}$$

where δ is the Dirac delta function. We implement this quantity of interest as a discrete delta function, such that the right-hand side for the adjoint problem takes the form

$$\frac{\partial J^h}{\partial \boldsymbol{u}^h} = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 \end{bmatrix}. \tag{2.31}$$

For this implementation, a mesh vertex is always placed at the spatial location $\boldsymbol{x}_0$, the QoI derivative vector is zeroed out, and we place a one in the row of the QoI derivative vector that corresponds to the i^{th} component of the solution at the vertex.

2.9.2 Integrated Solution Over a Sub-Domain

Next, we consider the integrated solution over a sub-domain $\Omega_0 \subset \Omega$, which can be expressed as

$$J(u) = \int_{\Omega_0} \frac{1}{n_c} \sum_{i=1}^{n_c} u_i \, \mathrm{d}\Omega. \tag{2.32}$$

Here, n_c denotes the number of components for the solution vector. As an example, Listing 2.4 demonstrates the Goal implementation for the QoI corresponding to the integrated displacement over a sub-domain.

2.9.3 Integrated von-Mises Stress Over a Sub-Domain

Finally, specifically for mechanics problems, we consider the evaluation of the von-Mises stress integrated over a sub-domain $\Omega_0 \subset \Omega$, given as

$$J(u) = \int_{\Omega_0} \sigma_{vm} \, \mathrm{d}\Omega, \tag{2.33}$$

Listing 2.4. Evaluation of the integrated displacement over a sub-domain.

```
1 template <typename ScalarT>
2 void AvgDisp<ScalarT>::at_point(
3     apf::Vector3 const&, double w, double dv) {
4   for (int i = 0; i < num_dims; ++i)
5     this->elem_value += u->val(i) * w * dv;
6   this->elem_value /= num_dims;
7 }
```

Listing 2.5. Evaluation of the integrated von-Mises stress over a sub-domain.

```
1 template <typename ScalarT>
2 void AvgVM<ScalarT>::at_point(
3     apf::Vector3 const&, double w, double dv) {
4   auto sigma = model->get_cauchy();
5   ScalarT vm = compute_von_mises<ScalarT>(sigma);
6   this->elem_value += vm * w * dv;
7 }
```

where the von-Mises stress σ_{vm} is defined as

$$\sigma_{vm} := \sqrt{\frac{3}{2}\boldsymbol{\sigma}'_{ij}\boldsymbol{\sigma}'_{ij}}. \tag{2.34}$$

Here summation over repeated indices is implied and $\boldsymbol{\sigma}' = \boldsymbol{\sigma} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})\boldsymbol{I}$ denotes the deviatoric part of the Cauchy stress tensor $\boldsymbol{\sigma}$. The von-Mises stress is often used in yield criterion for elastoplastic constitutive models, and is hence of particular interest for solid mechanics design applications.

We note that this funciton $J(u)$ has sources of nonlinearities from the deviatoric stress tensor $\boldsymbol{\sigma}'$ and further nonlinearites introduced by the definition of the von-Mises stress, which includes the square of deviatoric stress components and a square root operation. The linearization and implementation of this QoI, as required for adjoint-based error estimation, would be cumbersome at best without some kind of automated approach. In contrast, Listing 2.5 illustrates the simplicity of the relevant `C++` code that implements integration point contributions to this specific QoI in the Goal application.

2.10 Results

2.10.1 Poisson's Equation

As a first example, we investigate error estimation and mesh adaptation in Poisson's equation for the model problem

$$\begin{cases} -\nabla^2 u = f & \boldsymbol{x} \in \Omega, \\ u = 0 & \boldsymbol{x} \in \partial\Omega. \end{cases} \tag{2.35}$$

This model problem leads to the Galerkin finite element method: find $u^H \in \mathcal{V}^H$ such that $\mathcal{R}_g(w^H; u^H) = 0$ for all $w^H \in \mathcal{V}^H$. Here the residual $\mathcal{R}_g$ is defined as

$$\mathcal{R}_g(w^H; u^H) := (\nabla w^H, \nabla u^H) - (w^H, f), \tag{2.36}$$

and the space $\mathcal{V}^H$ is given by

$$\mathcal{V}^H := \{u^h \in H^1(\Omega) : u^H = 0 \text{ on } \partial\Omega \,,\, u^H|_{\Omega_e} \in \mathbb{P}^1\}. \tag{2.37}$$

Here Ω_e denotes an element in a decomposition of the domain Ω into n_{el} non-overlapping elements such that $\cup_{e=1}^{n_{el}} \Omega_e = \Omega$ and $\Omega_i \cap \Omega_j = \varnothing$ if $i \neq j$. Additionally, $\mathbb{P}^1$ denotes the space of piecewise linear polynomials.

The domain is chosen to be $\Omega := [-1, 1] \times [-1, 1] \setminus [-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$ as shown in Figure 2.3. The data is chosen to be $f = 1$ and we consider a point-wise QoI of the form $J(u) = \int_\Omega \delta(\boldsymbol{x} - \boldsymbol{x}_0) u \, \mathrm{d}\Omega$, where the point of interest $\boldsymbol{x}_0$ is chosen to be $\boldsymbol{x}_0 = (0.75, 0.75)$. This problem was initially studied in the reference [35], where the QoI was determined to have a reference value of $J(u) = 0.0334474 \pm 1e\text{-}7$. Presently, we demonstrate that our automated approach can reproduce the results for traditional adjoint-based error estimation found in [35].

Starting from the initial mesh shown in Figure 2.3, the steps:

$$\text{Solve Primal} \rightarrow \text{Solve Adjoint} \rightarrow \text{Estimate Error} \rightarrow \text{Adapt Mesh}$$

were performed seven times. The adaptive simulation was run using 4 MPI ranks. The mesh size field was set according to equation (2.28) so that the target number of elements is

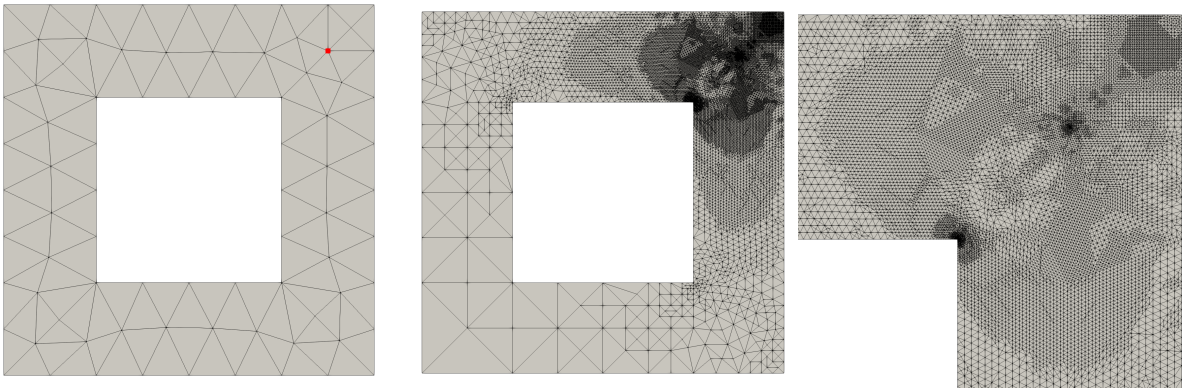

Fig. 2.3. Domain and initial mesh (left) for the Poisson's equation example with the QoI point indicated in red, final adapted mesh (middle), and a close up of the upper-right hand corner of the final adapted mesh (right).

twice that of the current mesh. Figure 2.3 also shows the final adapted mesh resulting from this procedure. We remark that the distribution of degrees of freedom in this mesh closely resembles the results obtained in reference [35].

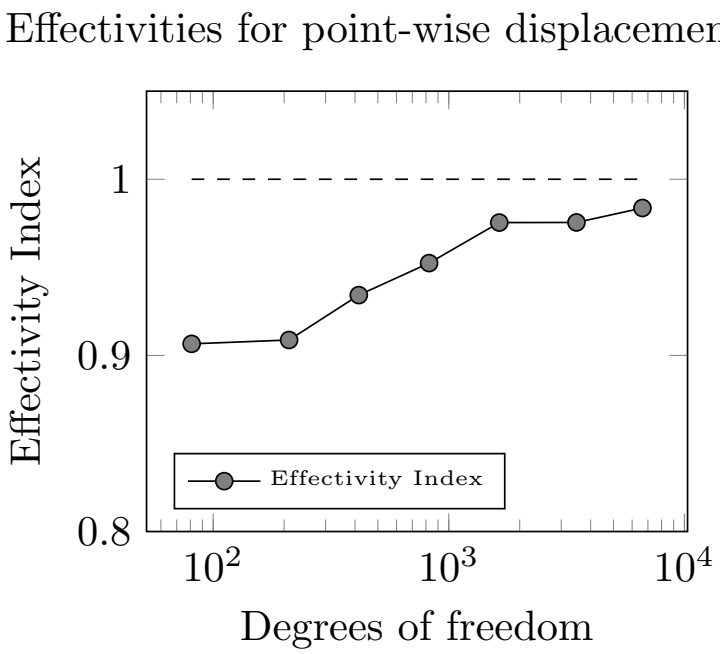

Fig. 2.4. Effectivity indices for the adaptive Poisson's equation example.

We expect this functional to converge at the rate $k = 2$, such that the scaling factor α used in the estimate (2.19) is given as $\alpha = \frac{3}{4}$. We consider the "exact error" $\mathcal{E} = J(u) - J(u^H)$

and the effectivity index $\mathcal{I} = \frac{\eta}{\mathcal{E}}$, where η is the estimate given by equation (2.19). Here we have placed quotations around the term exact error because we have only approximated the exact value of the QoI $J(u)$, and not truly recovered its exact value. Figure 2.4 plots the effectivity index $\mathcal{I}$ versus the number of degrees of freedom in the adaptive process. This plot demonstrates the ability of the error estimate to recover the "exact error" as $H \to 0$.

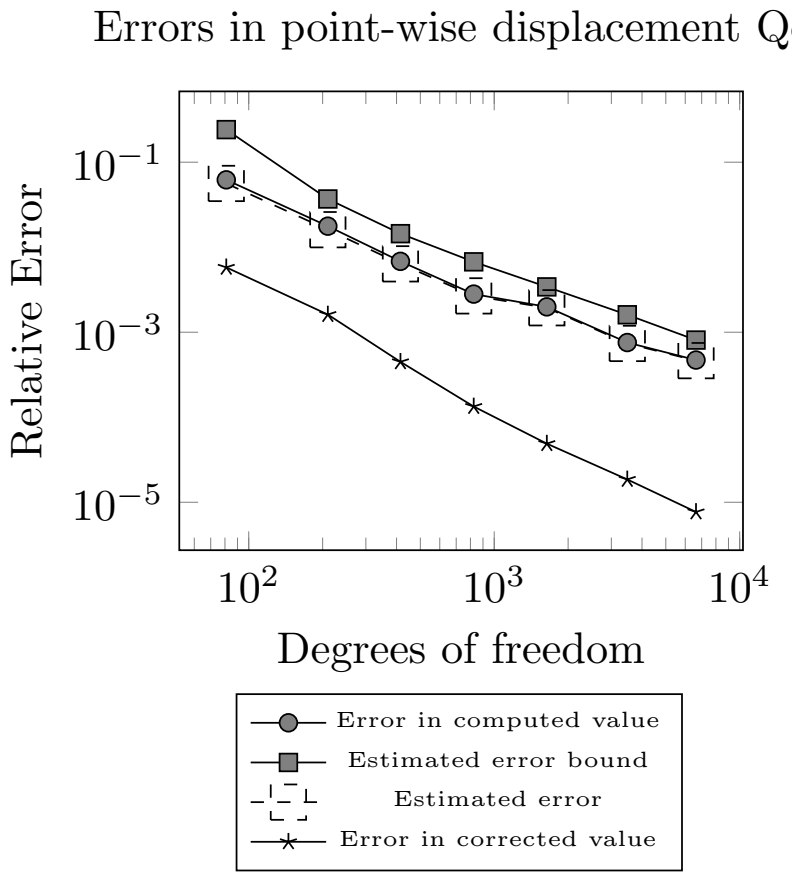

Fig. 2.5. Errors for the point-wise QoI for the adaptive Poisson's equation example.

Figure 2.5 displays the evolution of various errors during the adaptive process. The "exact error" $\mathcal{E}$ and the estimated error η are very close, as previously demonstrated by the effectivity index $\mathcal{I}$. Additionally, the approximated upper bound on the error $\hat{\eta}$ overestimates the error, but not to a large degree. This provides some indication that the correction indicators are effective in that they do not drastically overestimate error. Finally, we remark that an improved *corrected* QoI functional value can be computed as $J^*(u^H) = J(u^H) + \eta$.

Figure 2.5 demonstrates that this corrected value is nearly an order of magnitude more accurate than the computed functional value $J(u^H)$ during the adaptive process.

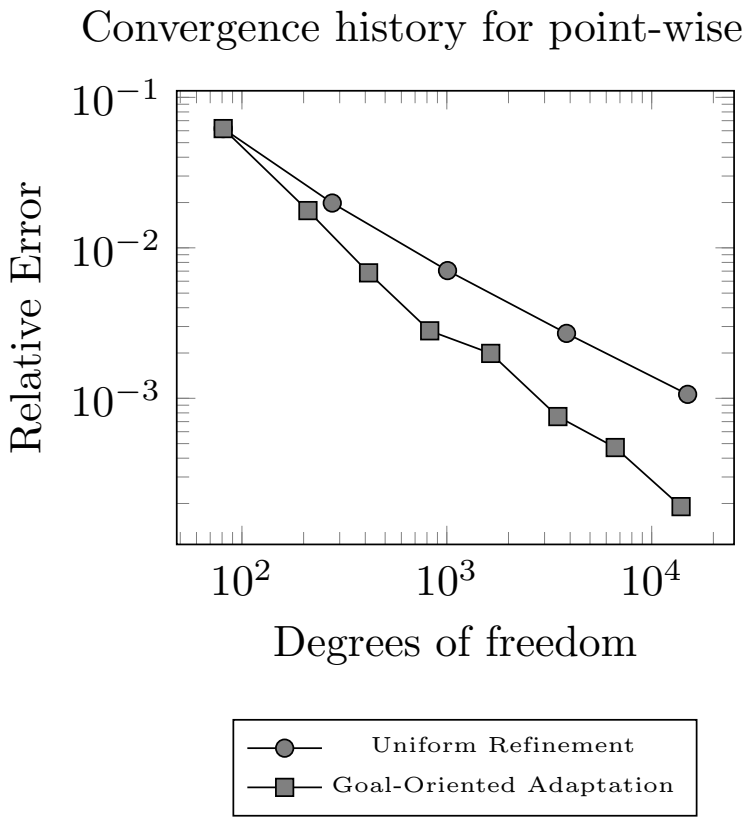

Fig. 2.6. Error convergence using uniform mesh refinement and adjoint-based error estimation for the adaptive Poisson's equation example.

Finally, Figure 2.6 demonstrates the evolution of the "exact error" for two adaptive strategies. The first strategy uniformly refines the mesh at each adaptive step and the second strategy performs the adjoint-based adaptive scheme developed in this work. We note that the error for the adjoint-based adaptive scheme converges faster than the uniform refinement scheme. Further, this convergence plot is consistent with the reference [35].

2.10.2 A Cell Embedded in a Matrix

Recently, the automated approach developed in this paper was applied to a stabilized mixed pressure-displacement finite element formulation [25] for the governing equations of finite deformation elasticity in a total Lagrangian setting [36] (see Chapter **??**). For two and

three dimensional problems in nonlinear elasticity, the automated approach was shown to effectively estimate the error and provide improved error convergence rates via adjoint-based mesh adaptation over uniform refinement.

In this section, the parallelization of a biomechanical application presented in the reference [36] is discussed. First, the governing equations are briefly reviewed. For mixed pressure-displacement formulations in the Goal application, the Galerkin residual is defined as:

$$\begin{aligned}&\mathcal{R}_g(\boldsymbol{W}^H;\boldsymbol{U}^H) := \\ &\int_\Omega \boldsymbol{P} : \nabla \boldsymbol{w}^H \,\mathrm{d}\Omega + \int_\Omega \left[\frac{p^H}{\kappa} - \frac{1}{2j}(j^2-1)\right] q^H \,\mathrm{d}\Omega - \int_{\partial\Omega_h} \boldsymbol{h}\cdot\boldsymbol{w} \,\mathrm{d}\Gamma,\end{aligned} \tag{2.38}$$

and the stabilization residual is defined as:

$$\mathcal{R}_\tau(\boldsymbol{W}^H;\boldsymbol{U}^H) := \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau_e (j\boldsymbol{F}^{-1}\boldsymbol{F}^{-T}) : (\nabla p^H \otimes \nabla q^H) \,\mathrm{d}\Omega. \tag{2.39}$$

Here, $\boldsymbol{F}$ is the deformation gradient, $j := \det(\boldsymbol{F})$, $\boldsymbol{h}$ is an applied traction over the boundary $\partial\Omega_h$, $\boldsymbol{P} := j\boldsymbol{\sigma}\boldsymbol{F}^{-T}$ is the first Piola-Kirchhoff stress tensor, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, n_{el} is the total number of elements in the mesh, and $\tau_e := \frac{c_0 H_e^2}{2\mu}$ is a mesh-dependent stabilization parameter, where c_0 is a non-negative stability constant, H_e denotes an element mesh size and μ denotes the bulk modulus. The Cauchy stress tensor is defined via a neo-Hookean constitutive relationship. The total solution vector is defined as $\boldsymbol{U}^H := [\boldsymbol{u}^H, p^H]$, where $\boldsymbol{u}^H$ corresponds to displacements and p^H corresponds to pressures. Similarly, the total weighting vector is defined as $\boldsymbol{W}^H := [\boldsymbol{w}^H, q^H]$, where $\boldsymbol{w}^H$ denotes a weighting function corresponding to displacements and q^H is a weighting function corresponding to pressures. For a complete exposition, we refer the reader to Chapter **??**.

We focus on a microglial cell with dimensions of about $20\mu m \times 20\mu m \times 20\mu m$ embedded in an extracellular matrix of dimension $100\mu \times 100\mu m \times 100\mu m$. The QoI is chosen to be a local integrated displacement $J(\boldsymbol{U}) = \int_{\Omega_0} \frac{1}{3}(u_x + u_y + u_z) \,\mathrm{d}\Omega$, defined over a box Ω_0 with dimensions $30\mu m \times 30\mu m \times 30\mu m$ that bounds the microglial cell. Figure 2.7 shows the geometry defining the microglial cell, the bounding box Ω_0, and the extracellular matrix. The shear modulus is defined as $\mu = 600$ Pa and Poisson's ratio is set to be $\nu = 0.4999$.

To drive the problem, traction boundary conditions are imposed along the surface of

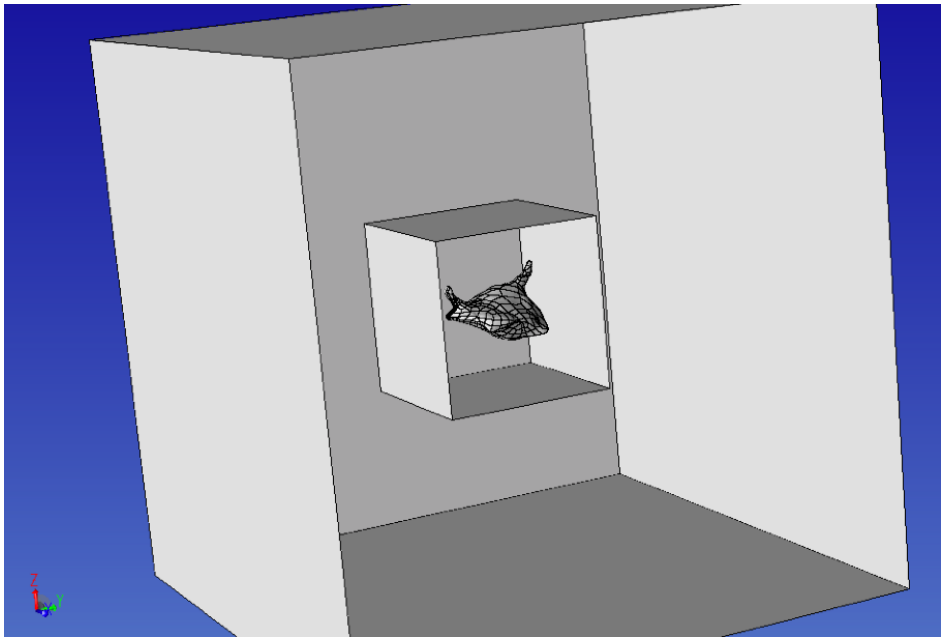

Fig. 2.7. Domains for the microglial cell example.

the microglial cell. The magnitude of the applied traction $\boldsymbol{h}$ is defined to be 10 times the distance to the center of the cell and its direction points inward towards the cell center. This traction is consistent with observed physical behavior [37]. Displacements $u_x = 0$, $u_y = 0$, and $u_z = 0$ are applied to the faces with constant minimum x-coordinate value, constant minimum y-coordinate value, and constant minimum z-coordinate value, respectively, to constrain rigid body rotations and translations.

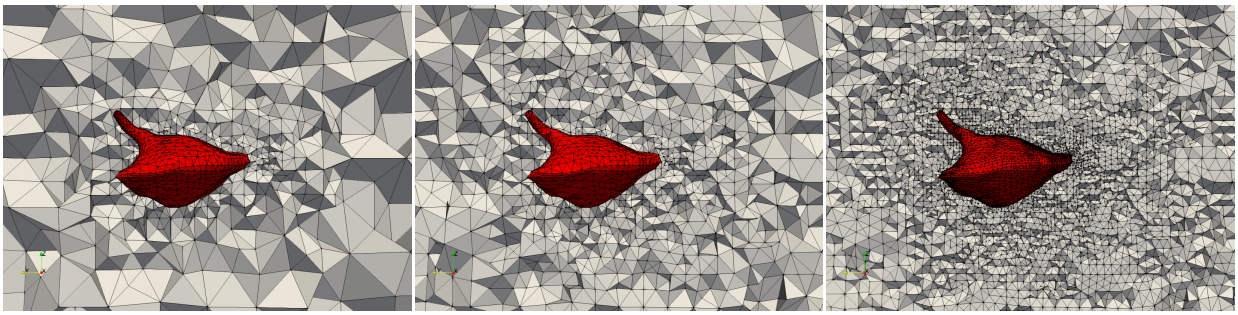

Fig. 2.8. A close-up of the initial mesh (left) the mesh after 5 adaptive iterations (center) and the final adapted mesh (right) for the microglial cell example.

Figure 2.8 demonstrates an initial mesh, which contains around $30,000$ degrees of

freedom. From this initial mesh, the steps

$$\text{Solve primal PDE} \rightarrow \text{Solve adjoint PDE} \rightarrow \text{Localize error} \rightarrow \text{Adapt mesh}$$

were successively performed 10 times. During the adapt stage, the mesh size field was set such that desired number of elements N in the output mesh is 1.5 times the number of elements in the previous mesh, according to equation (2.28). Figure 2.8 additionally demonstrates the adapted meshes obtained at the fifth and final adaptive iteration. In particular, both *coarsening* and *refinement* is performed during the adaptive iterations.

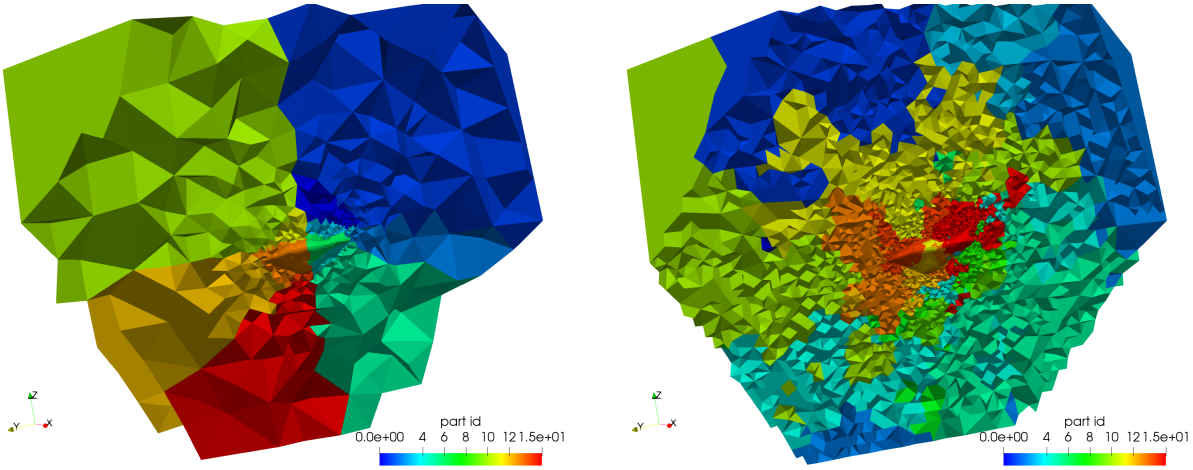

Fig. 2.9. The parallel mesh partitioning for the initial mesh (left) and the final adapted mesh (right) for the microglial cell example.

The problem was run using 16 MPI ranks. Figure 2.9 demonstrates the parallel partitioning for the initial mesh and for the final adapted mesh obtained after 10 adjoint-based adaptive iterations. To ensure partitioning quality, ParMA was utilized to guarantee the imbalance of vertices and elements across parallel partitions is no greater than 5%.

Figure 2.10 presents a breakdown of the total percentange of CPU time spent on each step in the adaptive analysis. For every adaptative iteration, the error localization (2.21) takes only a small percentage of the total CPU time, as it essentially amounts to an evaluation of the residual vector on the fine space. More interestingly, mesh adaptation initially accounts for about 20 percent of the total CPU time but decreases as the adaptive simulation progresses. This is explained by the fact that the initial adaptive iteration requires more work to optimally distribute the degrees of freedom for the functional QoI as compared to subsequent adaptive iterations. In addition to refinement and coarsening operations,

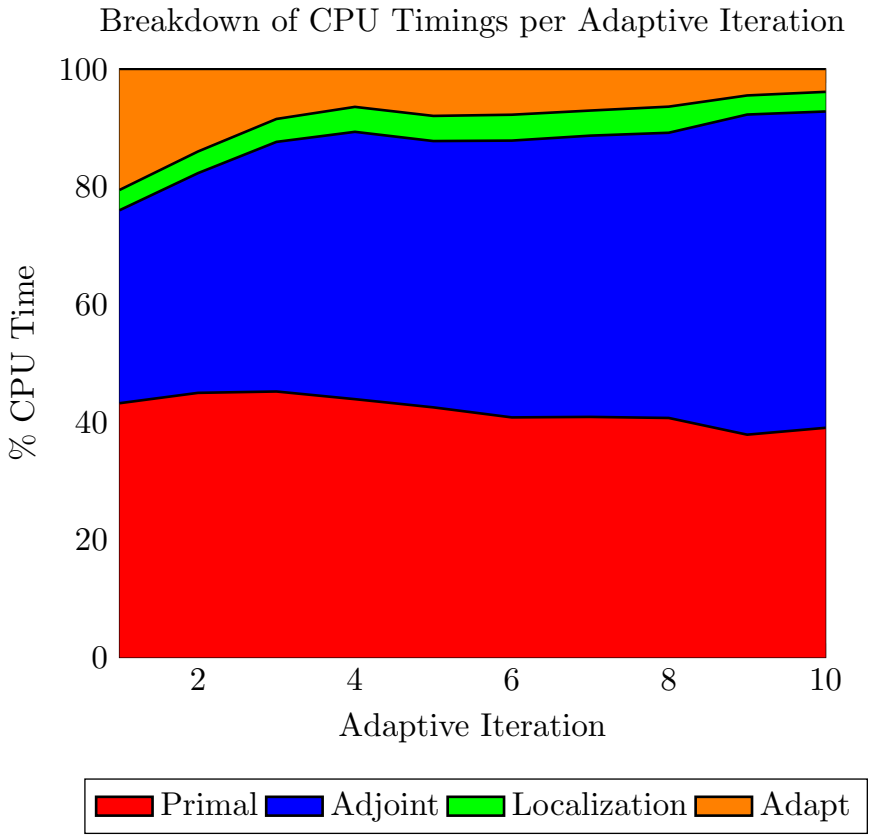

Fig. 2.10. Breakdown of the CPU time spent for each portion of the adaptive process for the microglial cell example.

the mesh adaptation step also performs *shape correction* to ensure elements are not too heavily skewed [17]. Finally, we note that the adjoint problem accounts for roughly 40 to 50 percent of the CPU time over the course of the adaptive simulation. While process of adjoint-based error estimation is not cheap for this example, we provide two justifying remarks. First, this problem required only 3 to 4 Newton iterations for each primal solve. For constitutive models with higher degrees of nonlinearity or for problems loaded to higher strains, it is not uncommon for Newton's method to converge in 7 to 10 iterations. In these scenarios, the relative cost of adjoint-based error estimation is not as extreme. Second, for this computational price, we have achieved very accurate error estimates as shown in Chapter **??**.

2.10.3 Elastoplasticity in an Array of Solder Joints

In this section, we investigate the utility of adjoint-based mesh adaptation for a thermomechanical analysis of an array of solder joints used in microelectronics fabrication. We consider a 6×6 array of solder joints sandwiched between two materials with distinct thermomechanical properties to model a portion of the process of 'flip-chip' manufacturing [38]. The full geometry is shown in Figure 2.11. We consider an elastoplastic constitutive model with a von-Mises yield surface and linear isotropic hardening, as given by Simo and Hughes [39] with a temperature correction for the stress tensor [40]. The top slab, solder joints, and bottom slab are modeled with the distinct material properties given in reference [38].

To drive the problem, the entirety of the domain is cooled from a reference temperature $T_{ref} = 393K$ to a resting temperature of $T_f = 318K$ in a single load step. The faces with minimum x, y, and z coordinate values were constrained to have zero displacements in the x, y, and z directions, respectively. As a QoI, we consider the integrated von-Mises stress given by equation (2.33) over three solder joints shown in yellow in Figure 2.11.

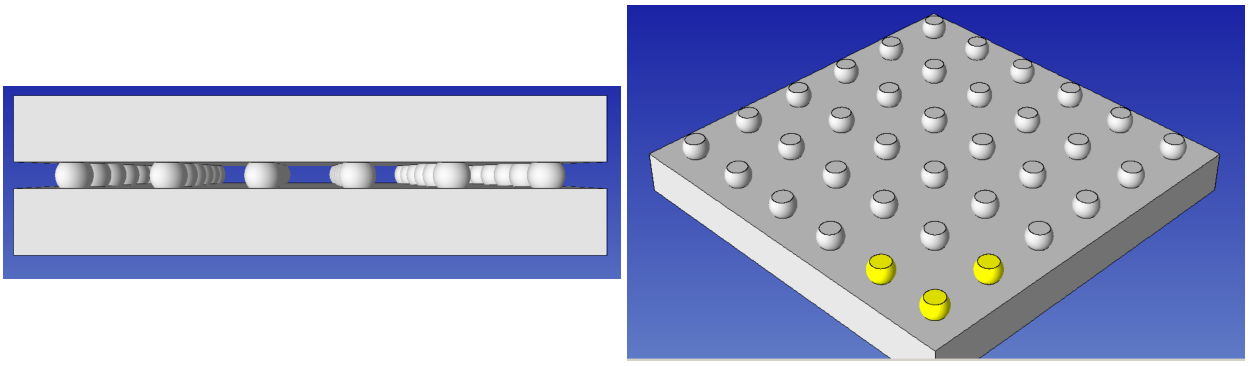

Fig. 2.11. The solder joint array geometry (left) and the geometric specification of the integrated von-Mises QoI (right).

The primal problem was solved on a sequence of uniformly refined meshes, starting with an initial mesh with about 1 million elements distributed over 16 MPI ranks, and finalizing with a mesh with over half a billion elements distributed over 8192 MPI ranks. For each solve, the work load for each mesh part (MPI rank) was held constant at approximately $70,000$ elements. Figure 2.12 demonstrates weak scaling timing results for various aspects of the primal solve. In particular, we remark that the assembly of the residual vector and Jacobian matrix scale well as the number of MPI ranks increases. The preconditioning routine shows a slight increase in time as the number of MPI ranks increases, but this increase is not drastic.

The time to solve the linear system, however, does not scale optimally. Improvements to parallel performance could likely be made by more finely tuning the preconditioning and linear solver routines for the specific problem, but this is outside the scope of the present work.

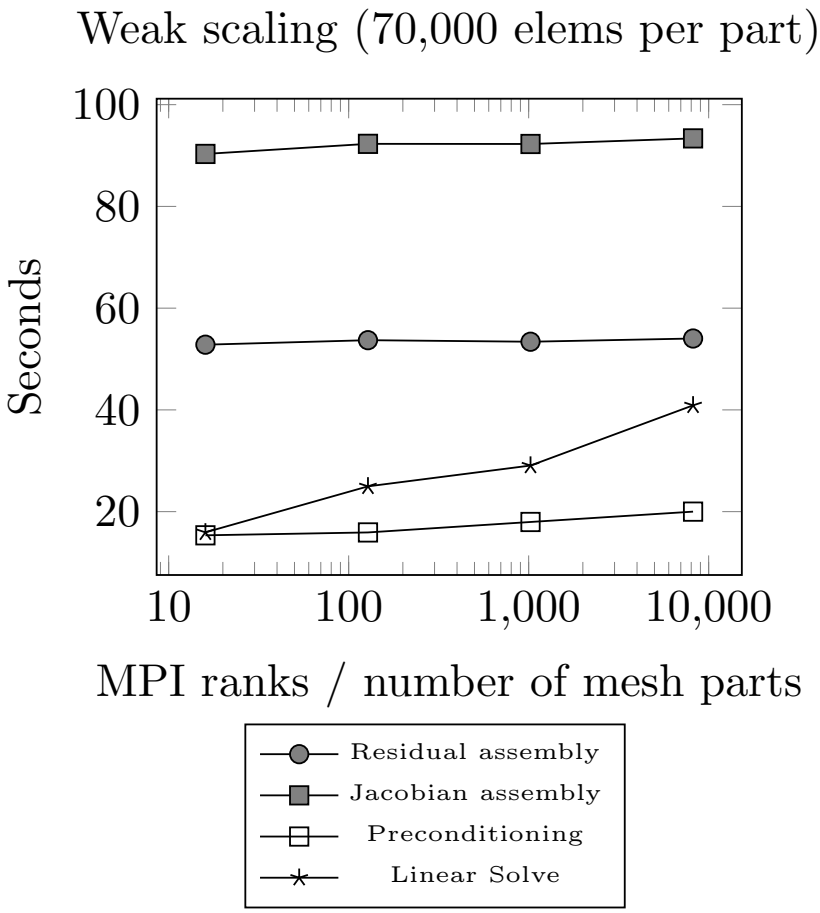

Fig. 2.12. Weak scaling for the Goal application.

The approximate QoI, $J^H(\boldsymbol{u}^H)$, was computed at each primal solve. Using the QoI evaluations from the finest three meshes, we performed Richardson extrapolation [41] to obtain a more accurate representation of the QoI. This value was given as $J(u) = 328.9$. We consider the extrapolated value to be the "true" QoI value and measure errors with respect to it. The expected convergence rate of the QoI is $k = 1$, which is confirmed by the Richardson extrapolation procedure.

From the same initial mesh used in the weak scaling study, we iteratively performed

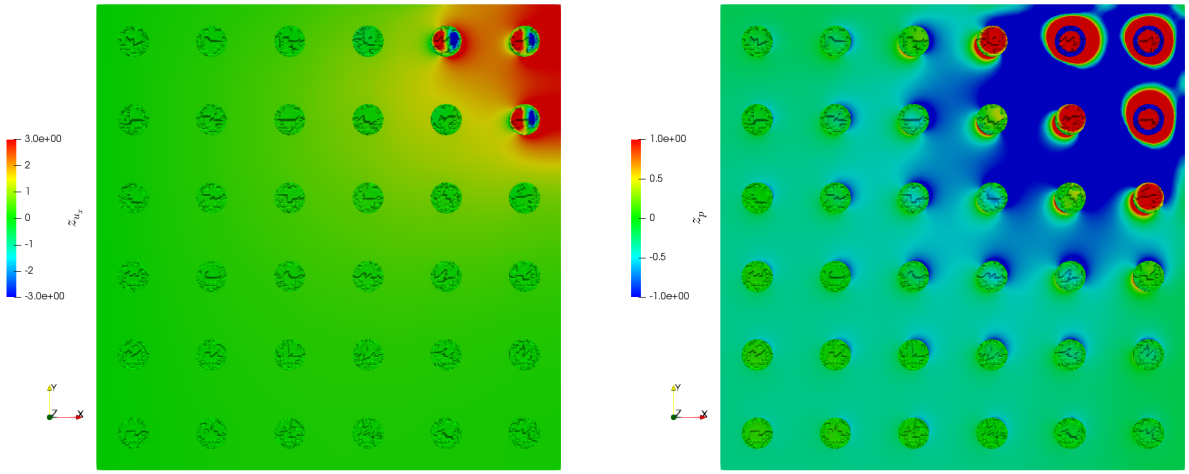

Fig. 2.13. The x-component of the adjoint displacement solution (left), and the pressure component of the adjoint solution (right).

the steps:

$$\text{Solve Primal} \rightarrow \text{Solve Adjoint} \rightarrow \text{Estimate Error} \rightarrow \text{Adapt Mesh}$$

with a restart after each mesh adaptation using 128, 256, and 512 MPI ranks, such that the output number of elements N in the adapted mesh was targeted to be 1 million, 2 million, and 4 million elements, respectively. Figure 2.13 shows different components of the adjoint solution obtained during the adjoint-based adaptive process. Figure 2.14 shows the spatial distribution of the error for the given QoI as computed by the adjoint-based error estimation process. Unsurprisingly, the majority of the error is localized to the area which geometrically defines the QoI. However, there are also contributions to the error from nearby solder joints that decrease as the distance from the 3 QoI solder joints increases. These additional contributions to the error are mostly gathered at the interface between solder joints and the underlying material slab, where von-Mises stress concentrations exist.

Figures 2.15 and 2.16 demonstrate the intial mesh used for the solder joint problem and the final adapted mesh obtained via adjoint-based adaptation. These figures clearly demonstrate that the 3 solder joints that define the QoI sub-domain are heavily refined, as expected. Additionally, notice that Figure 2.15 demonstrates that there is refinement at the left-most solder joint, which is not included in the geometric definition of the QoI. The adjoint-based error estimation procedure indicates that mesh must be refined in additional areas to accurately assess the QoI.

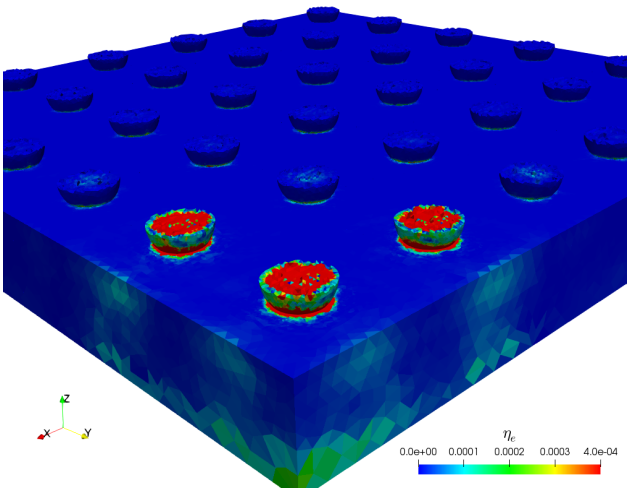

Fig. 2.14. The spatial distribution of errors as computed by adjoint-based error estimation for the solder joint array.

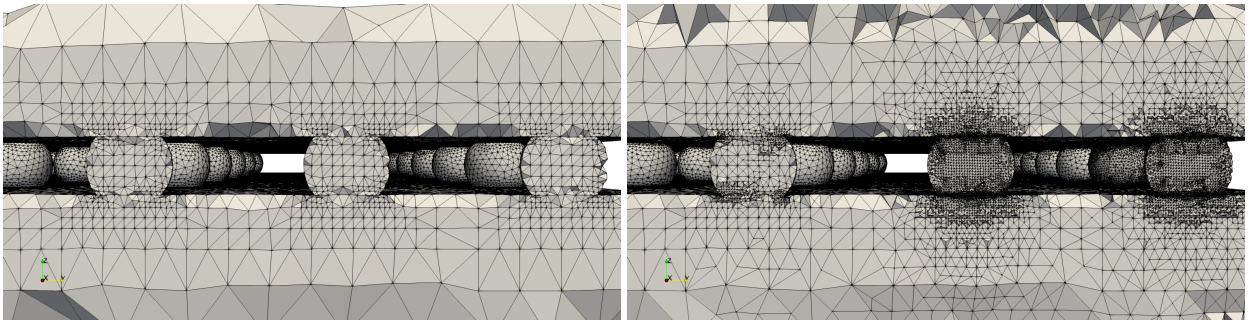

Fig. 2.15. Cross-sectional view of the initial mesh for the solder joint geometry (left) and the final adapted mesh (right).

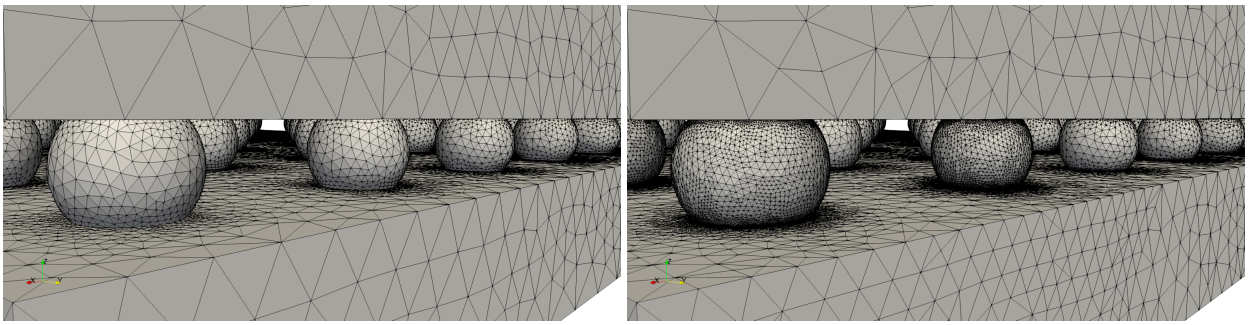

Fig. 2.16. The initial mesh for the solder joint geometry (left) and the final adapted mesh (right).

Figure 2.17 demonstrates the convergence history of the error in the functional QoI, as defined by the difference of the QoI obtained via Richardson extrapolation and the QoI approximated by the finite element solution. We compare the convergence for two adaptive

schemes, one achieved by successive uniform refinements of the mesh and the other achieved by adjoint-based error estimation. After 4 adaptive iterations, the adjoint-based adaptive procedure achieves nearly the same degree of accuracy as the uniform refinement procedure with two orders of magnitude fewer degrees of freedom.

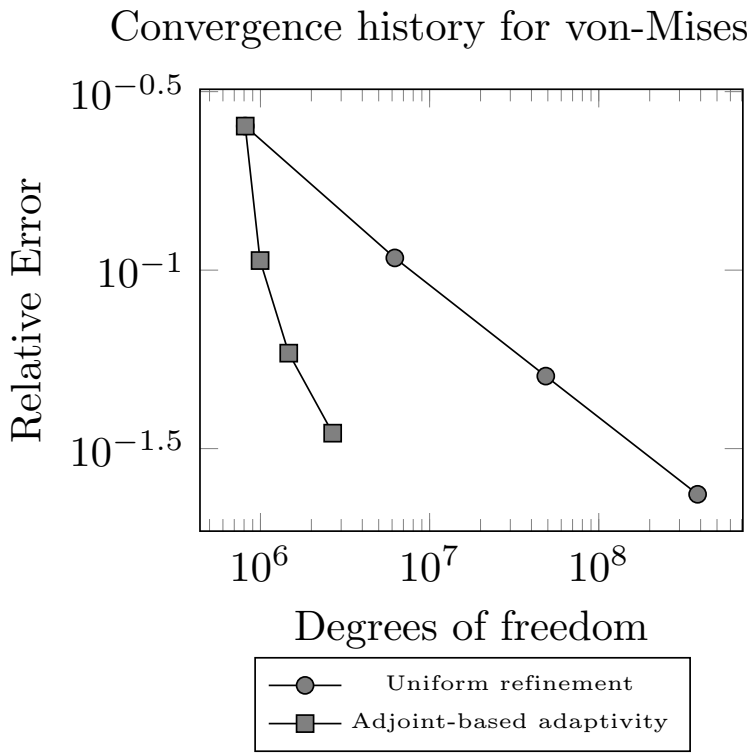

Fig. 2.17. Error convergence histories for the solder joint example problem with the integrated von-Mises stress QoI.

Finally, we remark that automated parallel adaptive workflows have been developed in reference [38]. As an avenue for future investigation, adjoint-based error estimation could be folded into these automated workflows. In particular, an automated primal analysis could be used to inform the actual selection of the QoI itself, which could then be accurately assessed using adjoint-based error estimation.

2.11 Conclusions

In this work, we have developed an automated approach for adjoint-based error estimation and mesh adaptation for execution on parallel machines. We have developed this

approach to be applicable to both Galerkin and stabilized finite element methods. To realize this approach, we have extended the concept of *template-based generic programming* for PDE models to include the automatic localization of error contributions using a partition of unity-based localization approach. We have demonstrated that this approach is effective for a variety of example applications, including nonlinear elasticity and elastoplasticity.

CHAPTER 3
A NON-UNIFORM REFINEMENT APPROACH FOR SOLVING ADJOINT PROBLEMS IN FUNCTIONAL ERROR ESTIMATION AND MESH ADAPTATION

3.1 Introduction

Adjoint-based error estimation [7], [9], [31]–[33], [42]–[46] is a tool used in numerical simulation to estimate the discretization error in physically meaningful output quantities. Combined with mesh adaptation, adjoint-based error estimation also provides the ability to control the discretization error. The process of adjoint-based error estimation relies on the introduction of an auxiliary *adjoint problem*, which is constructed using the solution to the original or *primal problem* of interest.

To obtain meaningful error estimates, the solution to the adjoint problem must be enriched in some manner. That is, it is necessary to obtain a representation of the adjoint solution in a richer space compared to the space used for the primal problem. Several strategies are commonly used to obtain an enriched adjoint representation. These approaches include solving the adjoint problem in a globally higher order polynomial space [28], solving the adjoint problem on a uniformly refined mesh [29], solving the adjoint problem in the same space as used for the primal problem and solving local patch-wise problems least squares problems [26] or performing patch-wise higher-order interpolation [7], and enriching the adjoint solution via variational multiscale methods [27].

Solving the adjoint problem in a globally higher order polynomial space or on a uniformly refined mesh is a computationally expensive proposition. On the other hand, solving the adjoint problem in the same space as used for the primal problem and enriching it via some local recovery operation may not be guaranteed to yield a more accurate adjoint solution. In this chapter, we propose a simple compromise and solve the adjoint problem on meshes obtained via non-uniform refinement.

The remainder of this chapter is structured as follows. First, we review adjoint-based error estimation for functional quantities using two discretization levels, a *coarse* space and a *fine* space. We then review three choices for the fine space obtained by refinement of

the mesh used for the coarse space. The first choice is the standard uniform refinement method, while the other two approaches form the fine space via non-uniform refinement. In each of these sections, we discuss the algorithm utilized to generate the fine space. We then investigate these three adjoint enrichment approaches when applied to examples in Poisson's equation and conclude with a summary of the results.

3.2 Error Estimation with Two Levels

3.2.1 Error Estimates

Following Venditti and Darmofal [31]–[33], we review output-based error estimation using two discretization levels. Let $\mathcal{V}^h$ and $\mathcal{V}^H$ denote finite dimensional spaces such that $\mathcal{V}^H \subset \mathcal{V}^h$. We refer to $\mathcal{V}^h$ and V^H as the *fine* space and the *coarse* space, respectively. Let $\boldsymbol{R}^H : \mathbb{R}^N \to \mathbb{R}^N$ denote the system of (potentially nonlinear) algebraic equations arising from a finite element discretization of a PDE on the coarse space $\mathcal{V}^H$, such that the solution vector $\boldsymbol{u}^H \in \mathbb{R}^N$ satisfies

$$\boldsymbol{R}^H(\boldsymbol{u}^H) = 0. \tag{3.1}$$

Similarly, let $\boldsymbol{R}^h : \mathbb{R}^n \to \mathbb{R}^n$ denote the system of algebraic equations arising from a finite element discretization of the same PDE on the fine space $\mathcal{V}^h$, such that

$$\boldsymbol{R}^h(\boldsymbol{u}^h) = 0, \tag{3.2}$$

where $\boldsymbol{u}^h \in \mathbb{R}^n$ is the solution vector on the fine space and $n > N$.

Let $J^H : \mathbb{R}^N \to \mathbb{R}$ denote a discrete representation of a physically meaningful functional quantity on the coarse space $\mathcal{V}^H$, and similarly let $J^h : \mathbb{R}^n \to \mathbb{R}$ denote the functional approximated on the fine space $\mathcal{V}^h$. Let $\boldsymbol{u}_H^h := \boldsymbol{I}_H^h \boldsymbol{u}^H$ denote the prolongation of the coarse space solution $\boldsymbol{u}^H$ onto the fine space $\mathcal{V}^h$ via interpolation, where $\boldsymbol{I}_H^h : \mathcal{V}^H \to \mathcal{V}^h$.

The functional evaluated on the fine space $J(\boldsymbol{u}^h)$ can be expanded in a Taylor series approximation about the prolonged coarse space solution $\boldsymbol{u}_H^h$ as

$$J^h(\boldsymbol{u}^h) = J^h(\boldsymbol{u}_H^h) + \left[\left.\frac{\partial J^h}{\partial \boldsymbol{u}^h}\right|_{\boldsymbol{u}_H^h}\right](\boldsymbol{u}^h - \boldsymbol{u}_H^h) + \dots \tag{3.3}$$

Similarly, the residual system of equations evaluated on the fine space $\boldsymbol{R}^h(\boldsymbol{u}^h)$ can be expanded about the prolonged coarse space solution $\boldsymbol{u}_H^h$ as

$$\boldsymbol{R}^h(\boldsymbol{u}^h) = \boldsymbol{R}^h(\boldsymbol{u}_H^h) + \left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right](\boldsymbol{u}^h - \boldsymbol{u}_H^h) + \dots \tag{3.4}$$

Using the governing relation (3.2) in the residual Taylor expansion (3.4) suggests a first order approximation for the discretization error between the spaces:

$$(\boldsymbol{u}^h - \boldsymbol{u}_H^h) \approx -\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right]^{-1} \boldsymbol{R}^h(\boldsymbol{u}_H^h). \tag{3.5}$$

Inserting the error approximation (3.5) into the functional Taylor expansion (3.3) suggests the error estimate:

$$J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h) \approx -\left[\frac{\partial J^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right]\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right]^{-1} \boldsymbol{R}^h(\boldsymbol{u}_H^h), \tag{3.6}$$

which can be re-written in terms of an *adjoint* variable $\boldsymbol{z}^h$ as

$$J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h) \approx -\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h), \tag{3.7}$$

where $\boldsymbol{z}^h \in \mathbb{R}^n$ is the solution to the so-called *adjoint problem* given by

$$\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right]^T \boldsymbol{z}^h = -\left[\frac{\partial J^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right]^T. \tag{3.8}$$

3.2.2 A Simple A-Priori Analysis

Consider that the functional of interest converges at the rate k, such that $J - J^h(\boldsymbol{u}_H^h) = cH^k$ and $J - J^h(\boldsymbol{u}^h) = ch^k$, where J is the exact value of the functional quantity of interest. Assume that the fine space is obtained via refinement of the coarse space. Consider the ratio

$$\frac{J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h)}{J - J^h(\boldsymbol{u}_H^h)} \approx \frac{-\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h)}{J - J^h(\boldsymbol{u}_H^h)} \tag{3.9}$$

which, as $H \to 0$, will tend towards [9]

$$\alpha := 1 - \left(\frac{h}{H}\right)^k. \tag{3.10}$$

Let η denote our approximation to the functional error $J - J^h(\boldsymbol{u}_H^h)$. Let $\mathcal{I}$ denote the effectivity index given by

$$\mathcal{I} = \frac{\eta}{J - J^h(\boldsymbol{u}_H^h)}. \tag{3.11}$$

We would like error estimates $\mathcal{E}$ that lead to effectivity indices of $\mathcal{I} = 1$ as $H \to 0$. To achieve this, we scale the two-level adjoint weighted residual estimate (3.7) by the inverse of the factor α, such that

$$\eta = -\frac{1}{\alpha}\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_{\boldsymbol{H}}^{\boldsymbol{h}}). \tag{3.12}$$

3.3 Choices for the Fine Space

3.3.1 Uniform Refinement

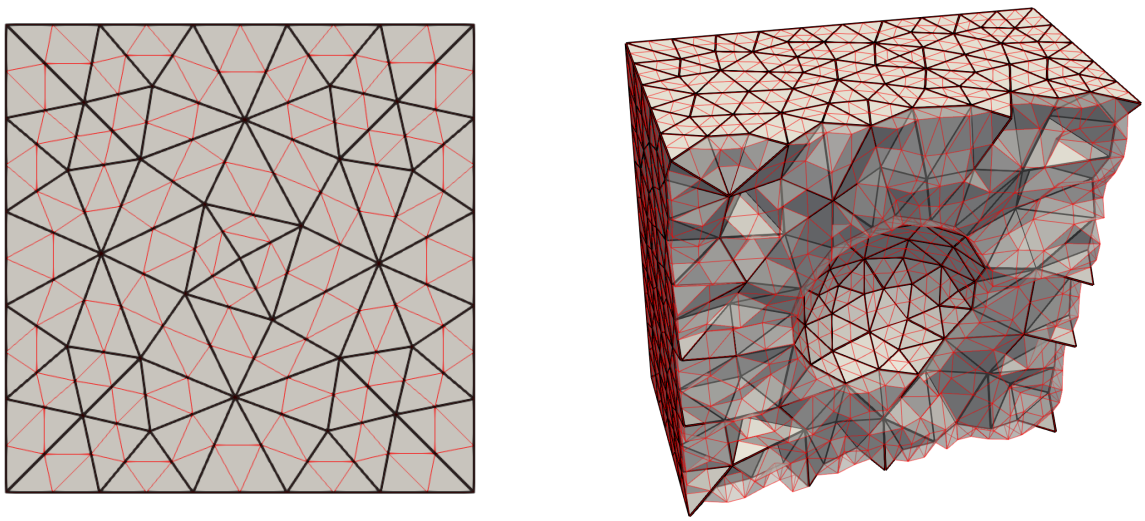

Fig. 3.1. Edges of a base mesh (black) and a nested mesh refined with the Unif scheme (red) in two dimensions.

We first consider the traditional approach of using a uniformly refined mesh to solve the adjoint problem. We refer to this approach as the Unif refinement approach. To perform

uniform refinement, every edge in the mesh is marked for refinement. The algorithm for uniform refinement is given in Algorithm 2. Figure 3.1 demonstrates an example of the UNIF refinement approach applied to a base mesh. For the uniform refinement approach, we naturally choose the ration $\frac{h}{H} = \frac{1}{2}$, leading to the scaling parameter $\alpha = 1 - \left(\frac{1}{2}\right)^k$.

Algorithm 2 Uniform refinement algorithm

for each edge e in mesh M **do**
    mark edge e for refinement.
end for

3.3.2 Long Edge Refinement

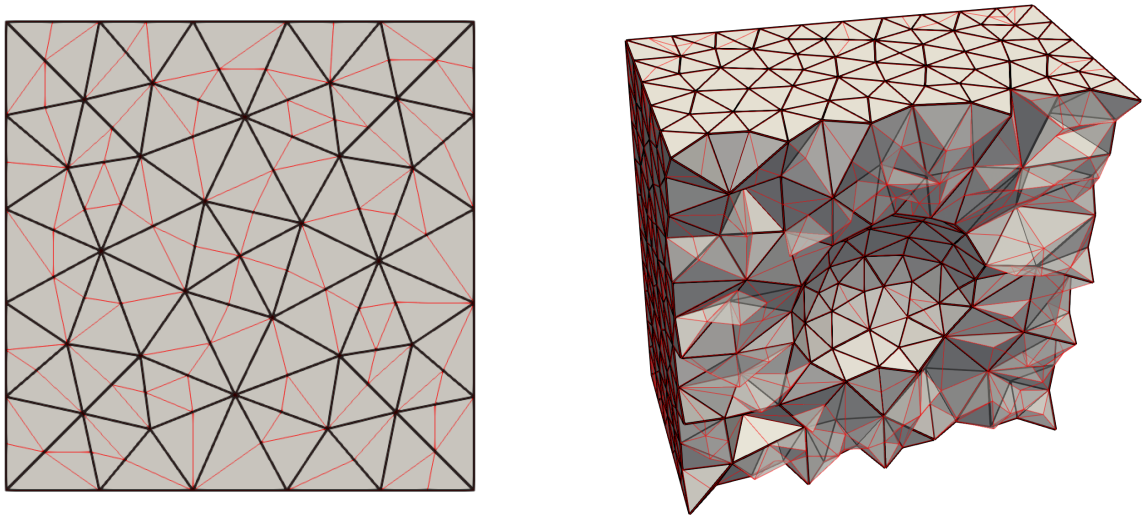

Fig. 3.2. Edges of a base mesh (black) and a nested mesh refined with the Long scheme (red) in two dimensions.

Next, we consider an adaptive scheme that marks the longest edge in each element for refinement. We refer to this scheme as the Long edge refinement scheme. The LONG edge refinement algorithm is outlined in Algorithm 3. Figure 3.2 illustrates the LONG edge refinement algorithm applied to a base mesh.

Note that, for the LONG edge refinement approach, some elements are split once while others are split multiple times. It follows then that there is no single global ratio $\frac{h}{H}$ of the fine mesh size to the coarse mesh size. Presently, we approximate this ratio by taking the

average of all ratios of nested element sizes to their parent element size, given by

$$\frac{h}{H} \approx \frac{1}{n_{el}} \sum_{e=1}^{n_{el}} \frac{h_e}{H_e}, \tag{3.13}$$

where n_{el} is the total number of elements in the nested mesh.

Algorithm 3 Long edge refinement algorithm

```
for each element el in mesh M do
    for each edge e in element el do
        if e is longest edge in el then
            mark edge e for refinement.
        end if
    end for
end for
```

3.3.3 Single Edge Refinement

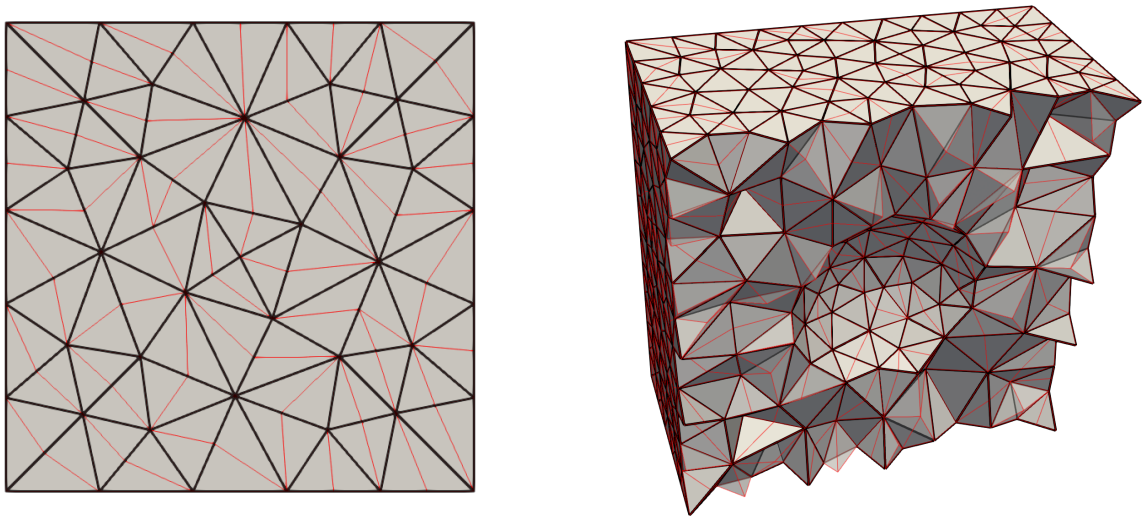

Fig. 3.3. Edges of a base mesh (black) and a nested mesh refined with the Single scheme (red) in two dimensions.

Finally, we consider a cheap refinement alternative to uniform refinement that attempts to only mark a single edge in each element for refinement. We refer to this approach as the SINGLE edge refinement approach. To perform single edge refinement, a traversal of all edges in the mesh is performed. During this traversal, the first edge encountered is marked for refinement and the elements adjacent to that edge are tagged as 'visited'. As the edges in the

mesh are traversed, each element adjacent to the edge is checked to see if it has already been encountered. If all adjacent elements have not been encountered, then the edge is marked for refinement. After this process has completed, some elements may be *isolated*, in that they have still not been marked as 'visited'. Thus, for each element remaining that has not been marked as 'visited', we mark the first edge adjacent to the element for refinement. The single edge refinement algorithm is illustrated in Algorithm 4. Figure 3.3 demonstrates a mesh resulting from the application of the single edge refinement scheme. For the SINGLE scheme, we again approximate the ratio $\frac{h}{H}$ with equation (3.13).

Algorithm 4 Single edge refinement algorithm

```
initialize all elements to be 'not visited'.
for each edge e in mesh M do
    let S be the set elements adjacent to edge e.
    if each element in S is 'not visited' then
        mark edge e for refinement.
        for each element el in S do
            mark element el as 'visited'.
        end for
    end if
end for
for each element el in mesh M do
    if el is marked as 'not visited' then
        let S be the edges adjacent to element el
        mark the first edge e in S for refinement
    end if
end for
```

3.4 Mesh Adaptation

3.4.1 Error Localization

It is necessary to localize contributions to the total error η to mesh entity level *correction indicators* to drive mesh adaptation. For finite volume and discontinuous Galerkin finite element methods, it is common to consider the discrete element-level adjoint weighted residuals of the form $\boldsymbol{z}_e^h \cdot \boldsymbol{R}_e^h$, where the subscript e denotes evaluations over elements. However, for continuous finite elements, this approach does not account for systematic inter-element cancellation [9], which could lead to a sub-optimal adaptive strategy.

Traditonally, for continuous Galerkin finite element methods, the error is localized by

integrating the residual by parts to recover strong form volumetric and jump contributions to the error over element interiors and boundaries, respectively. Presently, we utilize a localization strategy introduced by Richter and Wick [3] that proceeds by introducing a partition of unity ϕ_i, such that $\sum_i \phi_i = 1$, into the variational residual. In this localization, adjoint-weighted residual error information from neighboring elements is gathered to mesh vertices, leading to vertex-based correction indicators η_i, for $i = 1, 2, \ldots, n_{vtx}$. Here n_{vtx} denotes the number of vertices in the fine mesh. To obtain element-level correction indicators η_e, where $e = 1, 2, \ldots, n_{el}$, for the n_{el} elements in the space $\mathcal{V}^H$, we interpolate the vertex-based indicators η_i to element centers in the coarse mesh. While a full discussion of this localization procedure is outside of the scope of the present work, we refer readers to [3], [47] to demonstrate how this approach is utilized for Galerkin finite element methods and [36] to demonstrate how this approach is utilized for stabilized finite element methods.

3.4.2 Mesh Size Field

Once element-level correction indicators η_e have been obtained, we drive conforming mesh adaptation by specifying a *mesh size field*. For isotropic mesh adaptation, which we presently consider, this mesh size field defines the desired lengths of edges over the mesh. We utilize a mesh size field as described by Boussetta et al. [34] that attempts to equidistribute the error in an output adapted mesh with N target elements. From a high level, this size field will refine the mesh in areas of the domain that contribute strongly to the error in the functional and coarsen the mesh in areas of the domain that weakly contribute to the error in the functional.

Let p be the polynomial interpolant order for the chosen finite element method. In the subsequent results section, we consider only $p = 1$. We first define the global quantity G as

$$G = \sum_{e=1}^{n_{el}} (\eta_e)^{\frac{2d}{2p+d}}. \tag{3.14}$$

From this global quantity, we compute new element size H_e^{new} by scaling previous element sizes H_e according to the formula

$$H_e^{\text{new}} = \left(\frac{G}{N}\right)^{\frac{1}{d}} (\eta_e)^{\frac{-2}{2p+d}} H_e. \tag{3.15}$$

To ensure that mesh adaptation is being driven by accurate correction indicators and to prevent excessive coarsening and refinement in a single adaptive step, we additionally clamp new element sizes such that they are no smaller than one quarter and no greater than twice the previous element size,

$$\frac{1}{4} \leq \frac{H_e^{\text{new}}}{H_e} \leq 2. \tag{3.16}$$

Presently, we make use of the PUMI [10] software for mesh adaptation purposes. This software uses a sequence of edge splits, swaps, and collapses [17], [18] to locally modify the mesh to satisfy the input mesh size field.

3.5 Results

3.5.1 Effectivity Indices for Poisson's Equation

As a first example, we investigate the effectivity of the error estimate (3.12) for the model problem:

$$\begin{cases} -\nabla^2 u = f & \boldsymbol{x} \in \Omega, \\ u = 0 & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{3.17}$$

when using the Unif, Long, and Single approaches to solve the adjoint problem (3.8). The model problem leads to the Galerkin finite element method: find $u^H \in \mathcal{V}^H$ such that

$$(\nabla w^H, \nabla u^H) = (w, f) \quad \forall w^H \in \mathcal{V}^H, \tag{3.18}$$

where $(w, u) := \int_\Omega wu \, \mathrm{d}\Omega$ denotes the L^2 inner product over the space $\mathcal{V}^H$, defined as

$$\mathcal{V}^H := \{u^h \in H^1(\Omega) : u^H = 0 \text{ on } \partial\Omega \,,\, u^H|_{\Omega_e} \in \mathbb{P}^1\}. \tag{3.19}$$

Here Ω_e denotes an element in a decomposition of the domain Ω into n_{el} non-overlapping elements such that $\cup_{e=1}^{n_{el}} \Omega_e = \Omega$ and $\Omega_i \cap \Omega_j = \varnothing$ if $i \neq j$. Additionally, $\mathbb{P}^1$ denotes the space of piecewise linear polynomials.

We choose the domain $\Omega = [0,1] \times [0,1]$ and the data to be $f = 2\pi^2 \sin(\pi x) \sin(\pi y)$ such that the exact solution is $u(x,y) = \sin(\pi x) \sin(\pi y)$. We choose the functional quantity

to be $J(u) = \int_\Omega u \, \mathrm{d}\Omega$, which has the exact value $J(u) = \frac{4}{\pi^2}$. With the proposed finite element method, we expect the functional to converge at the rate $k = 2$, which we use to determine the scaling parameter α as given by equation (3.10).

The model problem was solved with mesh sizes $H = \{\frac{1}{5}, \frac{1}{10}, \frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}\}$. For each chosen mesh size, the discrete adjoint problem (3.8) was solved on fine spaces $\mathcal{V}^h$ generated by the UNIF, LONG, and SINGLE refinement schemes. An error estimate for the three schemes is then computed according to equation (3.12).

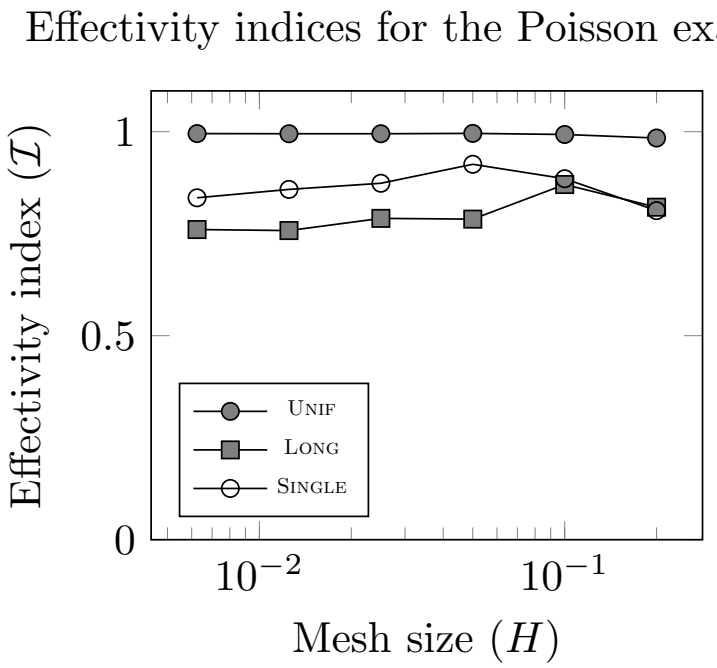

Fig. 3.4. Effectivity indices using the Unif, Long, and Single refinement schemes for the Poisson example problem.

Figure 3.4 plots the effectivity index (3.11) for each of the three schemes at each chosen mesh size. Effectivity indices for the baseline UNIF method approach 1 in the limit as $H \to 0$ as expected. The effectivity indices obtained using the two non-uniform refinement approaches, are less accurate and do not appear to be asymptotically correct. This is perhaps not surprising, as we have considered a bulk average for the ratio $\frac{h}{H}$ for these two schemes, as shown in Table 3.1.

However, even though the error estimates themselves obtained by the LONG and SINGLE schemes may not be suitable for application purposes, these schemes may still be suitable

Table 3.1. Approximated mesh size ratios for the Long and Single schemes for the first Poisson's equation example.

H	LONG : $\frac{h}{H}$	SINGLE : $\frac{h}{H}$
$\frac{1}{5}$	0.6344	0.8198
$\frac{1}{10}$	0.6323	0.8212
$\frac{1}{20}$	0.6325	0.8204
$\frac{1}{40}$	0.6490	0.8183
$\frac{1}{80}$	0.6477	0.8202
$\frac{1}{160}$	0.6467	0.8195

to drive mesh adaptation at a cheaper cost than the full UNIF approach. Figure 3.5 demonstrates the decrease in the total number of degrees of freedom for the adjoint problem for the LONG and SINGLE schemes as compared to the UNIF scheme. This motivates us to consider an adaptive example for Poisson's equation in the next section.

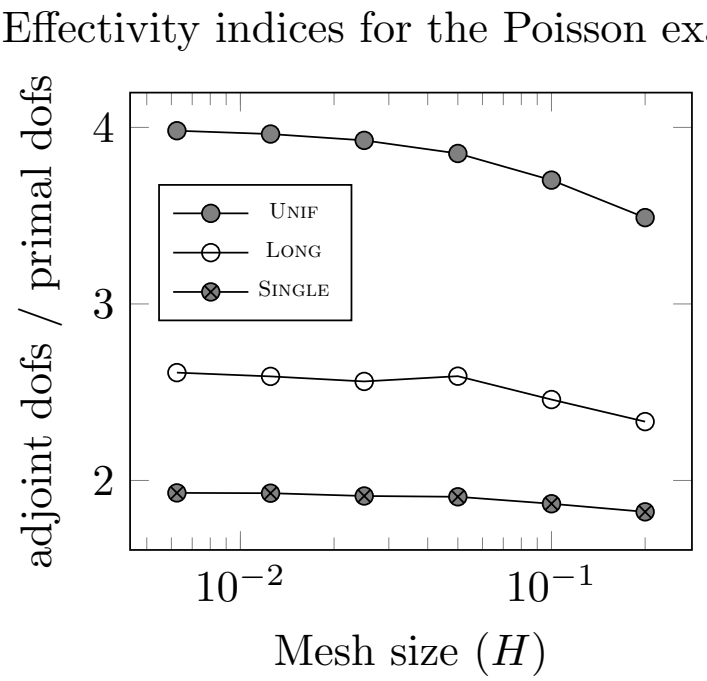

Fig. 3.5. Ratio of adjoint problem degrees of freedom to primal problem degrees of freedom using the Unif, Long, and Single refinement schemes for the Poisson example problem.

3.5.2 Mesh Adaptation for Poisson's Equation

In this example, we again consider the governing equations for Poisson's equation, as given in the previous section. However, we now choose the forcing function f to be $f = 1$ and the domain $\Omega := [-1, 1] \times [-1, 1] \setminus [-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. Further, we consider the point-wise quantity of interest $J(u) = \int_\Omega \delta(\boldsymbol{x} - \boldsymbol{x}_0) u \, \mathrm{d}\Omega$, where the point of interest is chosen to be $\boldsymbol{x}_0 = (0.75, 0.75)$. We again expect the functional to converge at the rate $k = 2$. The domain and point-wise QoI location are show in Figure 3.6. The value of the quantity of interest was determined to have a value of $J(u) = 0.0334473 \pm 1e\text{-}7$ in the reference [35].

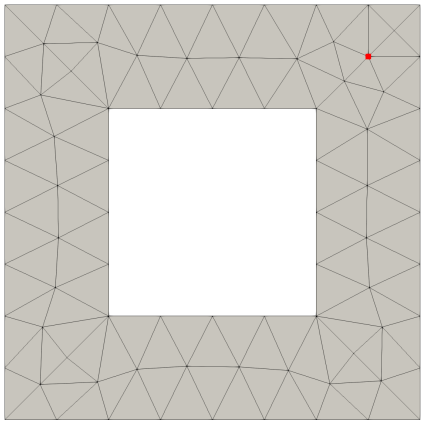

Fig. 3.6. Geometry and initial mesh used for the second Poisson's equation example with the point of interest shown in red.

We performed the steps:

$$\text{Solve Primal} \rightarrow \text{Solve Adjoint} \rightarrow \text{Estimate Error} \rightarrow \text{Adapt Mesh}$$

7 times, starting from the initial mesh shown in Figure 3.6. We solve the adjoint problem with three different methods on nested meshes obtained with the Unif, Long, and Single refinement schemes. At each adaptive step, the mesh size field was set according to equation (3.15), such that the target number of elements N is twice that of the current mesh.

Figure 3.7 illustrates the convergence history for the error $J(u) - J(u^H)$ for the three adaptive schemes obtained with the Unif (Goal Uniform), the Long (Goal Long), and the Single strategies, along with the error obtained by solving the primal problem with successively uniformly refined meshes (Uniform). The rate of convergence for the Unif scheme agrees with the reference [35]. Additionally, the error for both the Long and Single

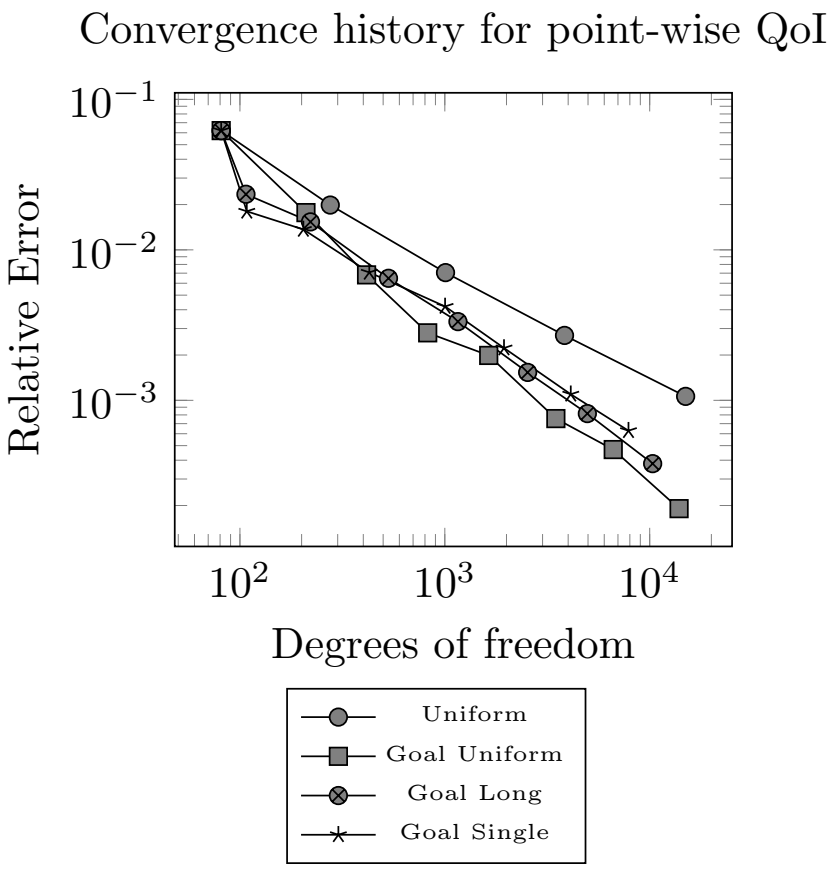

Fig. 3.7. Error evolution for adaptive schemes for the second Poisson's equation example.

schemes converges at a rate almost near the UNIF scheme.

Figure 3.8 illustrates the final adapted mesh obtained using the UNIF strategy to solve the adjoint problem. The distribution of degrees of freedom in this mesh closely resemembles the results obtained in reference [35]. However, using the LONG and SINGLE to solve the adjoint problem results in final adapted meshes that appear to be largely unsuitable for application analysis, as shown in Figures 3.9 and 3.10, even thought these meshes result in more accurate functional evaluations as compared to uniform refinement.

3.6 Conclusions and Outlook

We have developed two alternative approaches to uniform refinement for performing enriched adjoint solves in adjoint-based error estimation with two discretization levels. We

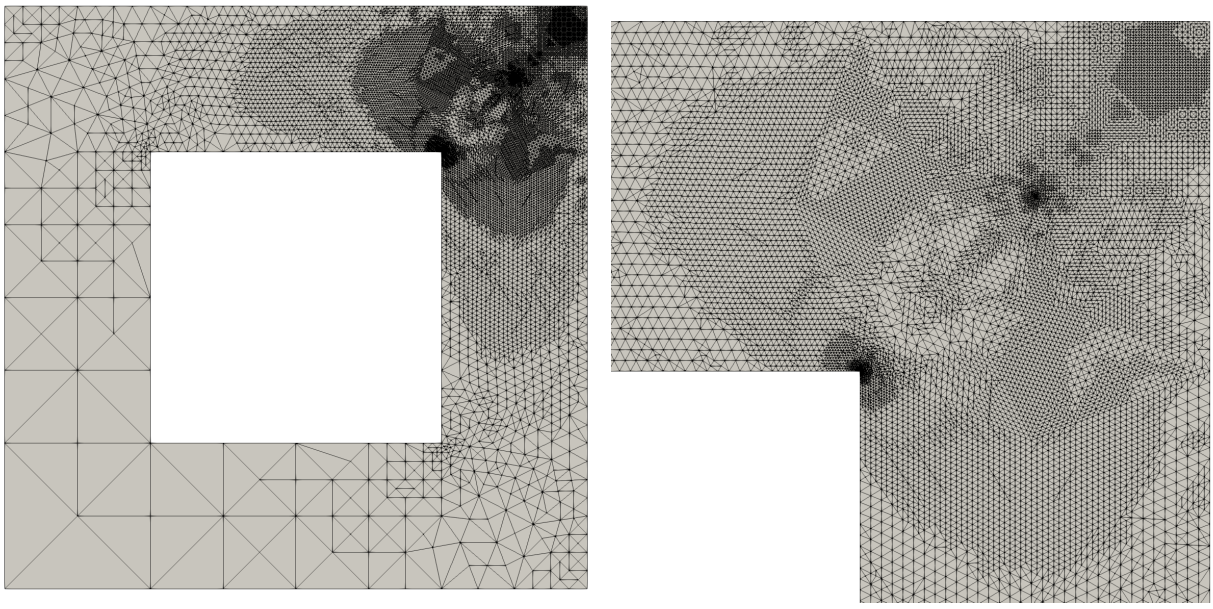

Fig. 3.8. The final adapted mesh using the Unif strategy to solve the adjoint problem (left) and a close-up of the upper right-hand corner of this mesh (right).

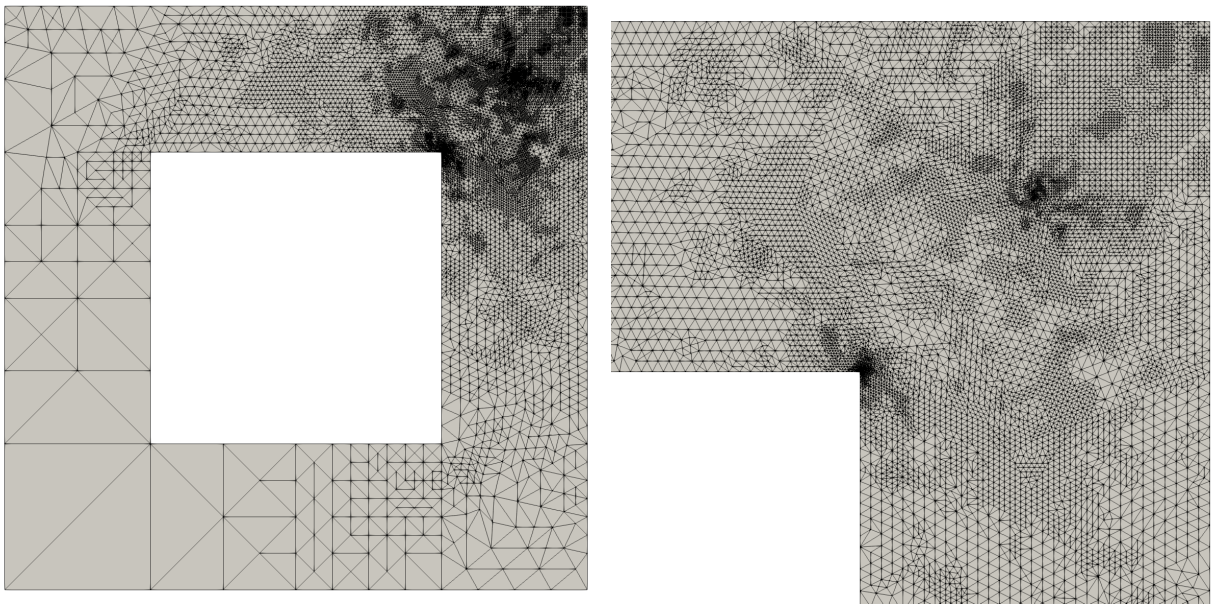

Fig. 3.9. The final adapted mesh using the Long strategy to solve the adjoint problem (left) and a close-up of the upper right-hand corner of this mesh (right).

have applied this approach to Poisson's equation. While the number of degrees of freedom for the adjoint solve for these two alternative approaches decreases significantly when

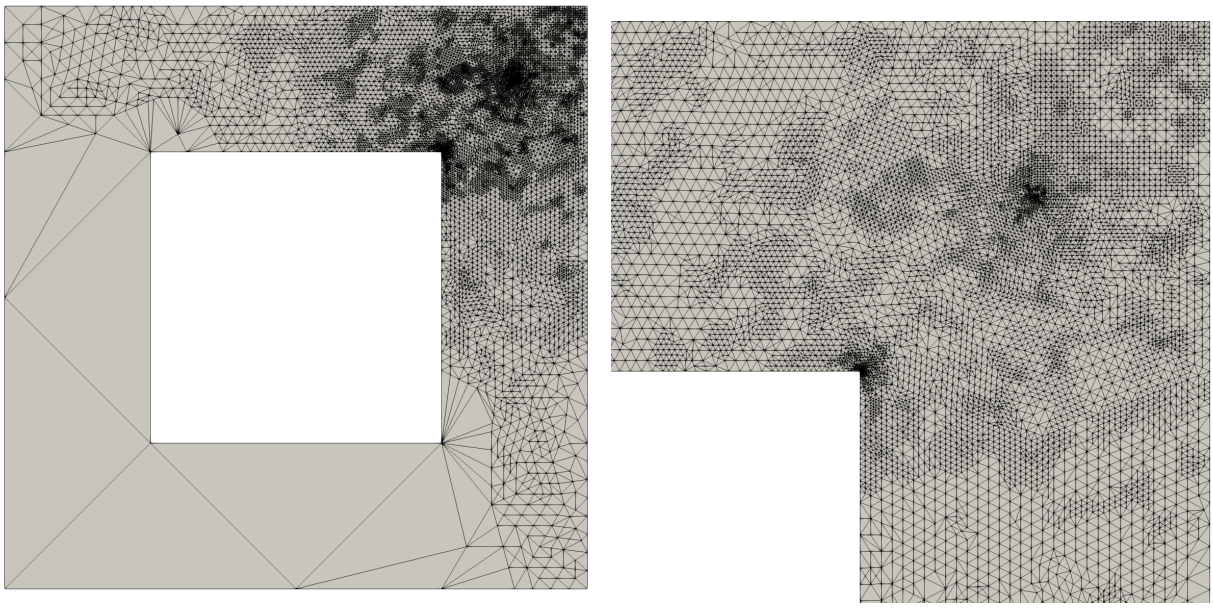

Fig. 3.10. The final adapted mesh using the Single strategy to solve the adjoint problem (left) and a close-up of the upper right-hand corner of this mesh (right).

compared to the more traditional approach of solving the adjoint problem on a uniformly refined mesh, the present outlook indicates that these approaches are not yet suitable for practical applications. That is, when performing adjoint-based error estimation with the two novel approaches, effectivity indices are not asymptotically correct. Additionally, the meshes obtained with adaptive adjoint-based analysis display qualitatively different features when compared to the uniform refinement approach.

It is possible that more accurate error estimates could be obtained by considering the total functional error as the sum of element-level contributions

$$J^h(u^h) - J^h(u_H^h) \approx \sum_{e=1}^{n_{el}} -\frac{1}{\alpha_e} \boldsymbol{z}_e^h \cdot \boldsymbol{R}_e^h(\boldsymbol{u}_H^h), \tag{3.20}$$

where we have replaced the approximated ratio (3.13) with the exact element-level ratio, $\alpha_e = 1 - \left(\frac{h_e}{H_e}\right)^k$. Here, the subscript e denotes the element-level contributions to the corresponding global quantity. Additionally, it is possible that more suitable meshes may be obtained during the adaptive process if a size field smoothing algorithm is utilized. We leave investigation into these areas as a suggestion for future work.

CHAPTER 4
OUTPUT-BASED ERROR ESTIMATION AND MESH ADAPTATION FOR VARIATIONAL MULTISCALE METHODS

4.1 Introduction and Motivation

Stabilized finite element methods have been used to effectively solve a wide variety of problems where standard Galerkin methods are known to be unstable. Among these problems are the advective-diffusive equations [48], [49], Stokes flow [50], [51], and the Navier-Stokes equations [52]–[54]. The variational multiscale (VMS) method, as developed by Hughes et al. [55], [56], provides a systematic approach to derive a stabilized finite element method. From a high level, the VMS approach decomposes the solution u to a partial differential equation (PDE) into *coarse-scale* components $\overline{u}$ and *fine-scale* components u', where the fine-scale solution is represented or approximated analytically.

A posteriori error estimation is a common tool to assess the accuracy and reliability of a finite element solution [57]. In the original developments of the VMS method, it was suggested that approximations to the fine-scale solution $u' = u - \overline{u}$ could be used to derive *a posteriori* error estimates [55]. Since then, numerous studies have utilized VMS techniques in the context of *a posteriori* error estimation. Hauke et al. [58], [59] investigated the using the fine-scale solution as an explicit error estimator in the context of advective transport problems. Masud et al. [60] derived explicit and implicit error estimates for the global discretization error for a mixed form of nearly incompressible elasticity, and then later extended these techniques to nonlinear elasticity formulations [61]. Larson and Målqvist [62] investigated approximating the fine-scale solution via local patch-wise problems, and derived an *a posteriori* error estimate for the solution in the energy norm for use in an adaptive finite element method.

Traditional *a posteriori* error estimates attempt to bound the error in a given norm. More recently developed duality-based *a posteriori* error estimates [9] seek to approximate the error in an output quantity that can be expressed as a functional $J(u)$. For example,

This chapter previously appeared as: B. N. Granzow, M. S. Shephard, and A. A. Oberai, "Output-based error estimation and mesh adaptation for variational multiscale methods." *Comput. Methods in Appl. Mechanics and Eng.*, vol. 322, pp. 331-459, Aug. 2017.

outputs corresponding to the lift or drag over an airfoil may be of primary interest for a numerical study. In general, output-based error estimates based on duality techniques require the solution of an auxiliary *dual problem*. In contrast, the original PDE of interest is referred to as the *primal problem*. Using the solution z to the dual problem, output error estimates are written, in part, as the product of two terms [7], [63]. The first term involves the residual $\mathcal{R}u^h$ of the primal PDE evaluated at the finite element solution. The second term, typically referred to as the weighting term, involves the difference $z - I^h z$ between the exact dual solution and the nodal interpolant of the exact dual solution onto the finite element space used to approximate the primal problem.

The exact dual z solution is generally unknown, and thus must be approximated to obtain functional error estimates. Note that if the dual solution is approximated in the same finite element space as used for the primal problem, then the weighting term in the output error estimate is identically zero. Thus some form of enrichment to the dual solution is required. Several enrichment procedures are commonly used. One approach is to approximate the exact dual solution in a globally richer finite element space than the one used for the primal problem. Another approach involves solving the dual problem using the same finite element space as used for the primal problem and enriching the dual solution via projection. Yet another approach involves using *a priori* estimates to bound the interpolation error in the dual solution.

In this chapter we propose a novel strategy for output-based error estimation, whereby the dual solution is enriched by the *fine-scale dual solution* z' using VMS techniques. This is achieved by the introduction of a general representation $\mathcal{E}_2$ for functional errors in VMS methods. Using this general representation, we introduce simple approximations to the fine and coarse scale solutions for both the primal and dual problems to derive an error estimate η_2.

We then seek to demonstrate the utility of this error representation in adaptive finite elements. This is achieved in part by comparison to a recently proposed explicit output-based error representation $\mathcal{E}_1$ that utilizes VMS techniques to entirely circumvent the solution of an auxiliary dual problem [64]. We prove that error estimates $\eta_1 \approx \mathcal{E}_1$ and $\eta_2 \approx \mathcal{E}_2$ based on this explicit error representation and the newly proposed VMS technique, respectively, are identical. However, we demonstrate that localization of the explicit error estimate η_1 is insufficient to drive mesh adaptation for *local* output quantities, whereas the estimate η_2

performs well.

The remainder of this chapter is structured as follows. We begin by presenting a review of the derivation of a VMS method for an abstract Dirichlet primal problem. Then we introduce simple approximations to the fine-scale solution u' and the coarse-scale solution $\overline{u}$ to obtain a computable numerical subgrid method for the primal problem. Next, we introduce an auxiliary dual problem to relate the output $J(u)$ to the primal problem. We then derive a VMS and subgrid method for the dual problem. Using the VMS methods for the primal and dual problems, we derive a general expression $\mathcal{E}_2$ for representing output errors in VMS methods, as well as the previously proposed error representation $\mathcal{E}_1$. Then, utilizing the approximations made for the primal and dual subgrid models, we derive error estimates $\eta_1 \approx \mathcal{E}_1$ and $\eta_2 \approx \mathcal{E}_2$ and demonstrate that these two quantities are identical. Next, we discuss the localization of these error estimates to element-level error indicators and how these indicators are used to drive mesh adaptation procedures. Then we investigate the effectivity of error estimates η_1 and η_2 for one and two dimensional example problems. We conclude by investigating the ability of the estimates η_1 and η_2 to drive mesh adaptation to accurately compute output quantities $J(u)$.

4.2 Review of VMS Methods

4.2.1 Model Problem

Let $\Omega \subset \mathbb{R}^d$ be an open bounded domain with smooth boundary $\partial\Omega$, where d is the number of spatial dimensions of the domain. Let $\mathcal{V}$ be a Hilbert space equipped with the norm $\|\cdot\|_{\mathcal{V}}$ and inner product $(\cdot,\cdot)_{\mathcal{V}}$ such that $\mathcal{V} = \{u \in H(\Omega) : u|_{\partial\Omega} = 0\}$, where $H(\Omega)$ is a Hilbert space defined over the domain Ω. Let $\mathcal{V}^*$ be the dual space of $\mathcal{V}$ and ${}_{\mathcal{V}}\langle\cdot,\cdot\rangle_{\mathcal{V}^*}$ denote the dual pairing between the two spaces given by ${}_{\mathcal{V}}\langle v,u\rangle_{\mathcal{V}^*} = \int_\Omega vu\,\mathrm{d}\Omega$. Let $\mathcal{L} : \mathcal{V} \to \mathcal{V}^*$ be a linear differential operator. Let $f \in \mathcal{V}^*$ be given data. We consider the abstract model problem of finding $u \in \mathcal{V}$ such that

$$\begin{cases} \mathcal{L}u = f, & \boldsymbol{x} \in \Omega, \\ u = 0, & \boldsymbol{x} \in \partial\Omega. \end{cases} \tag{4.1}$$

In B.1 we discuss extending this model problem to account for non-homogeneous Dirichlet and Neumann boundary conditions.

We define the residual operator $\mathcal{R} : \mathcal{V} \to \mathcal{V}^*$ as $\mathcal{R}u := f - \mathcal{L}u$, and we refer to (4.1) as the *primal problem*. The equivalent weak form of the primal problem can be stated as: find $u \in \mathcal{V}$ such that

$$ {}_{\mathcal{V}}\langle v, \mathcal{L}u \rangle_{\mathcal{V}^*} = {}_{\mathcal{V}}\langle v, f \rangle_{\mathcal{V}^*} \quad \forall\, v \in \mathcal{V}. \tag{4.2} $$

4.2.2 VMS Formulation

In this section, we review the foundations of the VMS method, as developed by Hughes et al. [55] and later refined by Hughes and Sangalli [56]. The basis of the method is the introduction of a sum decomposition of the solution u such that $u = \overline{u} + u'$. Here $\overline{u} \in \overline{\mathcal{V}}$ corresponds to the computable *coarse-scale* solution, while $u' \in \mathcal{V}'$ is associated with unresolved *fine-scales* of the solution. Further, it is assumed that the coarse-scale space $\overline{\mathcal{V}}$ and fine-scale space $\mathcal{V}'$ are closed subspaces of $\mathcal{V}$ and that $\overline{\mathcal{V}} \oplus \mathcal{V}' = \mathcal{V}$.

Using this sum decomposition, the weak form of the primal problem can be restated: find $\overline{u} + u' \in \mathcal{V}$ such that

$$ {}_{\mathcal{V}}\langle v, \mathcal{L}(\overline{u} + u') \rangle_{\mathcal{V}^*} = {}_{\mathcal{V}}\langle v, f \rangle_{\mathcal{V}^*} \quad \forall\, v \in \mathcal{V}, \tag{4.3} $$

which can be split into the two subproblems: find $\overline{u} + u' \in \mathcal{V}$ such that

$$ {}_{\mathcal{V}}\langle \overline{v}, \mathcal{L}\overline{u} \rangle_{\mathcal{V}^*} + {}_{\mathcal{V}}\langle \overline{v}, \mathcal{L}u' \rangle_{\mathcal{V}^*} = {}_{\mathcal{V}}\langle \overline{v}, f \rangle_{\mathcal{V}^*} \quad \forall\, \overline{v} \in \overline{\mathcal{V}}, \tag{4.4} $$

$$ {}_{\mathcal{V}}\langle v', \mathcal{L}\overline{u} \rangle_{\mathcal{V}^*} + {}_{\mathcal{V}}\langle v', \mathcal{L}u' \rangle_{\mathcal{V}^*} = {}_{\mathcal{V}}\langle v', f \rangle_{\mathcal{V}^*} \quad \forall\, v' \in \mathcal{V}'. \tag{4.5} $$

The goal of the VMS method is to eliminate the fine-scale solution u' from the first sub-problem (4.4) by expressing u' in terms of the coarse-scale solution $\overline{u}$. This results in a coarse-scale model involving only $\overline{u}$ that can then be solved numerically. However, the two sub-problems are not currently well-posed in terms of uniqueness. To ensure uniqueness, an optimality condition $\phi(\cdot)$ is chosen, for example $\phi(\cdot) = \| \cdot \|^2_{H^1(\Omega)}$ or $\phi(\cdot) = \| \cdot \|^2_{L^2(\Omega)}$. The

problem is then reposed in the optimal context:

$$\begin{aligned} \min_{\overline{u}} \quad & \phi(u-\overline{u}), \\ \text{s.t.} \quad & \begin{cases} \overline{u} \in \overline{\mathcal{V}}, \\ u' \in \mathcal{V}', \\ \mathcal{L}(\overline{u}+u') = f, \end{cases} \end{aligned} \tag{4.6}$$

The success of Hughes and Sangalli [56] is in showing that this optimality criteria defines a projector $\mathcal{P} : \mathcal{V} \to \overline{\mathcal{V}}$ onto the coarse-scale space such that $\mathcal{P}u' = 0$. Additionally, the projector $\mathcal{P}$ implicitly defines the fine-scale space $\mathcal{V}' = \{v \in \mathcal{V} : \mathcal{P}v = 0\}$. Using this projector, Hughes and Sangalli then show that the fine-scale solution can be analytically represented as:

$$u' = \underbrace{\left(\mathcal{G} - \mathcal{G}\mathcal{P}^T \left(\mathcal{P}\mathcal{G}\mathcal{P}^T\right)^{-1} \mathcal{P}\mathcal{G}\right)}_{\mathcal{G}'} \mathcal{R}\overline{u}, \tag{4.7}$$

where $\mathcal{G} = \mathcal{L}^{-1}$ is the classical Green's operator and $\mathcal{G}'$ is the so-called *fine-scale Green's operator*. Similarly, the fine-scale solution can be written in terms of the so-called *fine-scale Green's function* $g'(\boldsymbol{x};\boldsymbol{y})$ as

$$u'(\boldsymbol{y}) = \int_\Omega g'(\boldsymbol{x};\boldsymbol{y})(\mathcal{R}\overline{u})(\boldsymbol{x})\,\mathrm{d}\Omega_x, \tag{4.8}$$

where $g'(\boldsymbol{x};\boldsymbol{y})$ is defined by the operator $\mathcal{G}'$.

Let $\mathcal{L}^*$ be the adjoint operator of $\mathcal{L}$ such that

$${}_{\mathcal{V}}\langle v, \mathcal{L}u\rangle_{\mathcal{V}^*} = {}_{\mathcal{V}^*}\langle \mathcal{L}^* v, u\rangle_{\mathcal{V}} \quad \forall\, u, v \in \mathcal{V}. \tag{4.9}$$

We note that this equation represents the definition of the adjoint operator $\mathcal{L}^*$ and does not place any restriction on $\mathcal{L}$. When $\mathcal{L}$ is self adjoint, we have the identity $\mathcal{L}^* = \mathcal{L}$, otherwise $\mathcal{L}^*$ and $\mathcal{L}$ are different. However, equation (4.9) holds in either case.

Using the definition of the adjoint (4.9) and the representation of the fine-scale solution

(4.7), the first sub-problem (4.4) can be restated as: find $\overline{u} \in \overline{V}$ such that

$$ {}_{\mathcal{V}}\langle \overline{v}, \mathcal{L}\overline{u} \rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* \overline{v}, \mathcal{G}' \mathcal{R} \overline{u} \rangle_{\mathcal{V}} = {}_{\mathcal{V}}\langle \overline{v}, f \rangle_{\mathcal{V}^*} \quad \forall\, \overline{v} \in \overline{\mathcal{V}}. \tag{4.10} $$

We refer to this equation as the *continuous variational multiscale* formulation of the primal problem. For use in later derivations, we rewrite this formulation as:

$$ {}_{\mathcal{V}^*}\langle \mathcal{L}^* \overline{v}, \mathcal{G}' \mathcal{R} \overline{u} \rangle_{\mathcal{V}} = {}_{\mathcal{V}}\langle \overline{v}, \mathcal{R} \overline{u} \rangle_{\mathcal{V}^*} \quad \forall\, \overline{v} \in \overline{\mathcal{V}}, \tag{4.11} $$

recalling the definition of the primal residual operator $\mathcal{R}u := f - \mathcal{L}u$.

4.2.3 Subgrid Model

In practice, the continuous VMS model (4.10) is approximated by a finite element method. We refer to this approximate model as the *subgrid model*, as is common in the literature. The first step in this approximation is to choose the coarse-scale space to be a finite dimensional subspace, that is $\overline{\mathcal{V}} = \mathcal{V}^h$, and partition the domain Ω into n_{el} non-overlapping finite element subdomains Ω^e with boundaries $\partial\Omega^e$ for $e = 1, 2, \ldots, n_{el}$.

Next we note that an exact representation for the fine-scale Green's function $g'(\boldsymbol{x}; \boldsymbol{y})$ (and the fine-scale Green's operator $\mathcal{G}'$) is generally not obtainable. Thus, we must introduce an approximation for the fine-scale Green's function to accurately represent the fine-scale solution (4.8). To this end, we introduce the so-called *element-level Green's function* $g^e(\boldsymbol{x}; \boldsymbol{y})$, defined over element interiors as

$$ \begin{cases} \mathcal{L}^* g^e(\boldsymbol{x}; \boldsymbol{y}) = \delta(\boldsymbol{x} - \boldsymbol{y}), & \boldsymbol{x} \in \Omega^e, \\ \quad g^e(\boldsymbol{x}; \boldsymbol{y}) = 0, & \boldsymbol{x} \in \partial\Omega^e, \end{cases} \tag{4.12} $$

such that $g'(\boldsymbol{x}; \boldsymbol{y}) \approx g^e(\boldsymbol{x}; \boldsymbol{y})$.

Note that this approximation assumes that the fine-scale solution u' vanishes on element boundaries $\partial\Omega^e$. Hughes and Sangalli [56] show that, in one spatial dimension, the choice of an H^1 optimality condition $\phi = \| \cdot \|_{H^1}^2$ results in a completely *local* fine-scale Green's function. That is, when $d = 1$, the H^1 optimality condition ensures the equivalence of the fine-scale Green's function $g'(\boldsymbol{x}; \boldsymbol{y})$ and the element-level Green's function $g^e(\boldsymbol{x}; \boldsymbol{y})$. This result provides justification for approximating the fine-scale Green's function as the element-

level Green's function and motivates us to only consider $\phi(\cdot) = \|\cdot\|_{H^1}$ in this work.

As a further simplification, we approximate the element-level Green's function by it's *average* value over the element interior, and denote this value by $\boldsymbol{\tau}^e$, which can be expressed as:

$$\boldsymbol{\tau}^e = \frac{1}{\text{meas}(\Omega^e)} \int_{\Omega^e} \int_{\Omega^e} g^e(\boldsymbol{x}; \boldsymbol{y}) \, \mathrm{d}\Omega_x \, \mathrm{d}\Omega_y. \tag{4.13}$$

We note that more accurate approximations for the element-level Green's function can be made. For instance, Oberai and Pinsky [65] approximate the element-level Green's function by a polynomial scalar function involving *moments* of the element-level Green's function. We leave investigation into this area as a consideration for future work.

With this final approximation, the subgrid model can be written as: find $u^h \in \mathcal{V}^h$ such that

$$\mathcal{V}\big\langle v^h, \mathcal{L}u^h \big\rangle_{\mathcal{V}^*} + \mathcal{V}^*\big\langle \mathcal{L}^* v^h, \boldsymbol{\tau}^e \mathcal{R}u^h \big\rangle_{\mathcal{V}}^{\Omega'} = \mathcal{V}\big\langle v^h, f \big\rangle_{\mathcal{V}^*} \quad \forall\, v^h \in \mathcal{V}^h, \tag{4.14}$$

or equivalently as: find $u^h \in \mathcal{V}^h$ such that

$$\mathcal{V}^*\big\langle \mathcal{L}^* v^h, \boldsymbol{\tau}^e \mathcal{R}u^h \big\rangle_{\mathcal{V}}^{\Omega'} = \mathcal{V}\big\langle v^h, \mathcal{R}u^h \big\rangle_{\mathcal{V}^*} \quad \forall\, v^h \in \mathcal{V}^h, \tag{4.15}$$

where we have approximated the fine-scale solution over element interiors as

$$u'\big|_{\Omega^e} \approx \widetilde{u}'\big|_{\Omega^e} = \boldsymbol{\tau}^e \mathcal{R}u^h. \tag{4.16}$$

Here ${}_{\mathcal{V}^*}\langle \cdot, \cdot \rangle_{\mathcal{V}}^{\Omega'}$ denotes the 'broken' dual pairing over element interiors given by

$${}_{\mathcal{V}^*}\langle u, v \rangle_{\mathcal{V}}^{\Omega'} = \sum_{e=1}^{n_{el}} {}_{\mathcal{V}^*}\langle u, v \rangle_{\mathcal{V}}^{\Omega^e}, \tag{4.17}$$

where we have denoted the dual pairing over a single element interior as

$${}_{\mathcal{V}^*}\langle u, v \rangle_{\mathcal{V}}^{\Omega^e} = \int_{\Omega^e} uv \, \mathrm{d}\Omega. \tag{4.18}$$

We emphasize that the approximations made to the fine-scale solution imply that the subgrid model (4.14) is an approximation to the continuous VMS formulation (4.10), which

in turn implies that the subgrid solution u^h is an approximation to the coarse-scale solution $\overline{u}$. With this in mind, we can express the exact solution u as

$$u = u^h + \widetilde{u}' + \widetilde{u} \tag{4.19}$$

where $\widetilde{u} = (\overline{u} - u^h) + (u' - \widetilde{u}')$ represents the approximation errors in the coarse and fine-scale solutions.

Remark 1. The finite element method is derived from the weak form of a partial differential equation that has been integrated by parts. Thus instead of the duality pairing used in equation (4.14) it makes use of the L_2 inner product, and the finite element subgrid model derived from the variational multiscale method is given by:

$$A(v^h, u^h) + (\mathcal{L}^* v^h, \tau^e \mathcal{R} u^h)_{\Omega'} = l(v^h) \quad \forall\, v \in \mathcal{V}. \tag{4.20}$$

where $A(\cdot,\cdot)$ is the bilinear form associated with the operator $\mathcal{L}$, $(\cdot,\cdot)_{\Omega'}$ is the broken L_2 inner product defined on element interiors, and $l(\cdot)$ is the linear functional associated with the forcing function f.

4.3 The Dual Problem

4.3.1 Abstract Problem

Let $J(u) : \mathcal{V} \to \mathbb{R}$ be a linear functional corresponding to a physically meaningful quantity of interest. We assume that $J(u)$ can be expressed as

$$J(u) = {}_{\mathcal{V}^*}\langle q, u \rangle_{\mathcal{V}}\,, \tag{4.21}$$

where $q \in \mathcal{V}^*$. Following standard duality-based approaches for *a posteriori* error estimation [7], [9], [63], [66] we introduce the *dual problem* : find $z \in \mathcal{V}$ such that

$${}_{\mathcal{V}^*}\langle \mathcal{L}^* z, v \rangle_{\mathcal{V}} = {}_{\mathcal{V}^*}\langle q, v \rangle_{\mathcal{V}} \quad \forall\, v \in \mathcal{V}. \tag{4.22}$$

The equivalent strong form of the dual problem can be written as: find $z \in \mathcal{V}$ such that

$$\begin{cases} \mathcal{L}^* z = q, & \boldsymbol{x} \in \Omega, \\ z = 0, & \boldsymbol{x} \in \partial\Omega. \end{cases} \tag{4.23}$$

We define the residual operator $\mathcal{R}^* : \mathcal{V} \to \mathcal{V}^*$ of the dual problem as $\mathcal{R}^* z := q - \mathcal{L}^* z$.

4.3.2 VMS Formulation

If the primal problem necessitates the use of numerical stabilization, it is also likely that solving the dual problem (4.22) with a Galerkin finite element method will yield spurious oscillations in the dual solution [8]. To prevent non-physical behavior in the dual solution, we also solve the dual problem with a VMS method. Cyr et al. [8] call this approach the *stabilization of the adjoint*. This is in contrast to deriving a dual problem directly from the primal subgrid model (4.14), which Cyr et al. refer to as the *adjoint of the stabilization*.

Let $\overline{\mathcal{V}}_d$ and $\mathcal{V}'_d$ be closed subspaces of $\mathcal{V}$. Obtaining a VMS formulation for the dual problem proceeds in exactly the same manner as the primal problem. First a sum decomposition of the dual solution z is assumed such that $z = \overline{z} + z'$, where $\overline{z} \in \overline{\mathcal{V}}_d$ and $z' \in \mathcal{V}'_d$. The weak form of the dual problem (4.22) is then written as two sub-problems, whose solutions are uniquely determined by an optimality condition $\phi_d(\cdot)$ imposed on the coarse-scale solution $\overline{z}$. As with the primal model, we will only consider the H^1 optimality condition $\phi_d(\cdot) = \| \cdot \|^2_{H^1}$. However, we note that one could potentially choose different optimality conditions for both the primal and dual problems. We leave investigation into this area as an open research topic.

The optimality condition defines a projector $\mathcal{P}_d : \mathcal{V} \to \overline{\mathcal{V}}_d$ onto the coarse-scale subspace such that $P_d z' = 0$, and this projector implicitly defines the fine-scale subspace as $\overline{\mathcal{V}}_d = \{v \in \mathcal{V} \,:\, \mathcal{P}_d v = 0\}$. If we let $\mathcal{G}_d$ denote the classical Green's operator for the dual problem, such that $\mathcal{G}_d = (\mathcal{L}^*)^{-1}$, then the fine-scale dual solution can be represented as:

$$z' = \underbrace{\left(\mathcal{G}_d - \mathcal{G}_d \mathcal{P}_d^T \left(\mathcal{P}_d \mathcal{G}_d \mathcal{P}_d^T\right)^{-1} \mathcal{P}_d \mathcal{G}_d\right)}_{\mathcal{G}'_d} \mathcal{R}^* \overline{z}, \tag{4.24}$$

where $\mathcal{G}'_d$ is the *dual fine-scale Green's operator*. Similarly, the fine-scale solution can be

written in terms of the *dual fine-scale Green's function*, $g'_d(\boldsymbol{x};\boldsymbol{y})$, as:

$$z'(\boldsymbol{y}) = \int_\Omega g'_d(\boldsymbol{x};\boldsymbol{y})(\mathcal{R}^*\overline{z})(\boldsymbol{x})\,\mathrm{d}\Omega_x, \tag{4.25}$$

where $g'_d(\boldsymbol{x};\boldsymbol{y})$ is defined by the operator $\mathcal{G}'_d$. Using this representation of the fine-scale dual solution, the *continuous variational multiscale* formulation of the dual problem is stated as: find $\overline{z} \in \overline{\mathcal{V}}_d$ such that

$$_{\mathcal{V}^*}\langle \mathcal{L}^*\overline{z}, \overline{v}\rangle_{\mathcal{V}} + {}_{\mathcal{V}}\langle \mathcal{G}'_d\mathcal{R}^*\overline{z}, \mathcal{L}\overline{v}\rangle_{\mathcal{V}^*} = {}_{\mathcal{V}^*}\langle q, \overline{v}\rangle_{\mathcal{V}} \quad \forall\, \overline{z} \in \overline{\mathcal{V}}_d. \tag{4.26}$$

Recalling the definition of the dual residual operator $\mathcal{R}^* := q - \mathcal{L}^*$, we can rewrite equation (4.26) as:

$${}_{\mathcal{V}}\langle \mathcal{G}'_d\mathcal{R}^*\overline{z}, \mathcal{L}\overline{v}\rangle_{\mathcal{V}^*} = {}_{\mathcal{V}^*}\langle \mathcal{R}^*\overline{z}, \overline{v}\rangle_{\mathcal{V}} \quad \forall\, \overline{z} \in \overline{\mathcal{V}}_d. \tag{4.27}$$

4.3.3 Subgrid Model

To derive a corresponding subgrid model to the VMS formulation of the dual problem (4.26), we will assume that the coarse-scale spaces for the primal and dual problem are chosen to be the same, such that $\overline{\mathcal{V}} = \overline{\mathcal{V}}_d$. Additionally, we will consider approximations made using the same finite dimensional subspace $\overline{\mathcal{V}} = \mathcal{V}^h$ and discretization as used for the primal subgrid model. We first approximate the dual fine-scale Green's function $g'_d(\boldsymbol{x};\boldsymbol{y})$ using the *dual element-level Green's function*, defined over element interiors as

$$\begin{cases} \mathcal{L}g^e_d(\boldsymbol{x};\boldsymbol{y}) = \delta(\boldsymbol{x}-\boldsymbol{y}), & \boldsymbol{x} \in \Omega^e, \\ g^e_d(\boldsymbol{x};\boldsymbol{y}) = 0, & \boldsymbol{x} \in \partial\Omega^e, \end{cases} \tag{4.28}$$

such that $g'_d(\boldsymbol{x};\boldsymbol{y}) \approx g^e_d(\boldsymbol{x};\boldsymbol{y})$.

We further approximate the fine-scale dual solution z' by writing it as the product of a scalar function τ^e_d times the dual residual operating on the coarse-scale solution. The scalar function is given as:

$$\tau^e_d = \frac{1}{\mathrm{meas}(\Omega^e)} \int_{\Omega^e}\int_{\Omega^e} g^e_d(\boldsymbol{x};\boldsymbol{y})\,\mathrm{d}\Omega_x\,\mathrm{d}\Omega_y. \tag{4.29}$$

The dual subgrid model can then be written as: find $z^h \in \mathcal{V}^h$ such that

$$ {}_{\mathcal{V}^*}\big\langle \mathcal{L}^* z^h, v^h \big\rangle_{\mathcal{V}} + {}_{\mathcal{V}}\big\langle \tau_d^e \mathcal{R}^* z^h, \mathcal{L} v^h \big\rangle_{\mathcal{V}^*}^{\Omega'} = {}_{\mathcal{V}^*}\big\langle q, v^h \big\rangle_{\mathcal{V}} \quad \forall\, v^h \in \mathcal{V}^h, \tag{4.30} $$

or equivalently as: find $z^h \in \mathcal{V}^h$ such that

$$ {}_{\mathcal{V}}\big\langle \tau_d^e \mathcal{R}^* z^h, \mathcal{L} v^h \big\rangle_{\mathcal{V}^*}^{\Omega'} = {}_{\mathcal{V}^*}\big\langle \mathcal{R}^* z^h, v^h \big\rangle_{\mathcal{V}} \quad \forall\, v^h \in \mathcal{V}^h, \tag{4.31} $$

where we have approximated the fine-scale dual solution over element interiors as

$$ z'\big|_{\Omega^e} \approx \widetilde{z}'\big|_{\Omega^e} = \tau_d^e \mathcal{R}^* z^h. \tag{4.32} $$

We note that the exact dual solution can be expressed as the sum

$$ z = z^h + \widetilde{z}' + \widetilde{z}, \tag{4.33} $$

where $\widetilde{z} = (\overline{z} - z^h) + (z' - \widetilde{z}')$ represents the approximation errors in the coarse and fine-scale solutions.

Remark 2. The finite element version of the dual subgrid model corresponding to equation (4.30) is written using the L_2 inner product as:

$$ A(z^h, v^h) + (\tau_d^e \mathcal{R}^* z^h, \mathcal{L} v^h)_{\Omega'} = J(v^h) \quad \forall\, v \in \mathcal{V}^h, \tag{4.34} $$

where $A(\cdot,\cdot)$ is the bilinear form associated with the operator $\mathcal{L}$, and $(\cdot,\cdot)_{\Omega'}$ is the broken L_2 inner product defined on element interiors.

4.4 Error Estimation

In this section, we develop a general framework for output-based error estimation in VMS methods. We first develop two error representations for output quantities in the continuous VMS setting. Next, we discuss the role of the approximations made in both the primal and dual subgrid models. Finally, we introduce two error estimates for output quantities. We prove that the error estimates are identical. However, we demonstrate the superiority of one estimate over the other in the context of error localization needed to drive

mesh adaptation.

4.4.1 Continuous VMS Error Representations

Proposition 1. *For any solution $u = u' + \overline{u}$ to the continuous VMS formulation* (4.10)*, we have the error representation*

$$\mathcal{E}_1 = J(u) - J(\overline{u}) = J(u'). \tag{4.35}$$

Proof. The result follows directly from the linearity of $J(\cdot)$ and the sum decomposition $u = u' + \overline{u}$. □

This error representation is used by Hauke and Fuster [64] to derive an explicit *a posteriori* error estimate for output quantities. The error estimate only involves an approximation $\widetilde{u}'$ to the fine-scale solution u' and completely avoids the solution of a dual problem. However, when q is chosen to be a *local* forcing function for the dual problem (e.g. a function which is non-zero only over a subdomain of the total domain), error estimates derived from this representation fail to provide useful information when they are localized to the element level. Such error localization is critical to drive mesh adaptation and is discussed in detail later.

Proposition 2. *For any solutions $u = u' + \overline{u}$ to the continuous VMS formulation* (4.10) *and $z = z' + \overline{z}$ to the continuous dual VMS formulation* (4.26)*, we have the error representation*

$$\mathcal{E}_2 = J(u) - J(\overline{u}) = {}_{\mathcal{V}}\langle \mathcal{G}_d' \mathcal{R}^* \overline{z}, \mathcal{R}\overline{u} \rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* \overline{z}, \mathcal{G}' \mathcal{R} \overline{u} \rangle_{\mathcal{V}} \,. \tag{4.36}$$

Proof.

$$
\begin{aligned}
J(u) - J(\overline{u}) &= {}_{\mathcal{V}^*}\langle q, u\rangle_{\mathcal{V}} - {}_{\mathcal{V}^*}\langle q, \overline{u}\rangle_{\mathcal{V}} && \text{by (4.21)} \\
&= {}_{\mathcal{V}^*}\langle \mathcal{L}^* z, u\rangle_{\mathcal{V}} - {}_{\mathcal{V}^*}\langle \mathcal{L}^* z, \overline{u}\rangle_{\mathcal{V}} && \text{by (4.22)} \\
&= {}_{\mathcal{V}}\langle z, \mathcal{L}u\rangle_{\mathcal{V}^*} - {}_{\mathcal{V}}\langle z, \mathcal{L}\overline{u}\rangle_{\mathcal{V}^*} && \text{by (4.9)} \\
&= {}_{\mathcal{V}}\langle z, f\rangle_{\mathcal{V}^*} - {}_{\mathcal{V}}\langle z, \mathcal{L}\overline{u}\rangle_{\mathcal{V}^*} && \text{by (4.2)} \\
&= {}_{\mathcal{V}}\langle z, \mathcal{R}\overline{u}\rangle_{\mathcal{V}^*} && \text{by definition, linearity} \\
&= {}_{\mathcal{V}}\langle z', \mathcal{R}\overline{u}\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}}\langle \overline{z}, \mathcal{R}\overline{u}\rangle_{\mathcal{V}^*} && \text{by definition, linearity} \\
&= {}_{\mathcal{V}}\langle z', \mathcal{R}\overline{u}\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^*\overline{z}, \mathcal{G}'\mathcal{R}\overline{u}\rangle_{\mathcal{V}} && \text{by (4.11)} \\
&= {}_{\mathcal{V}}\langle \mathcal{G}_d'\mathcal{R}^*\overline{z}, \mathcal{R}\overline{u}\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^*\overline{z}, \mathcal{G}'\mathcal{R}\overline{u}\rangle_{\mathcal{V}}\,. && \text{by (4.24)}
\end{aligned}
$$

□

This error representation suggests a general approach to output-based error estimation for VMS methods, where the only approximation made to this point is that the coarse-scale subspace for the primal and dual problems are equal, such that $\overline{\mathcal{V}} = \overline{\mathcal{V}}_d$. To derive computable error estimates, exact representations or approximations must be known for the fine-scale Green's operators and the coarse-scale solutions for both the primal and dual problems.

4.4.2 Subgrid Model Error Representations

We now derive error representations that arise by introducing the approximations made in the primal and dual subgrid models.

Proposition 3. *For any solutions u to the primal model* (4.2) *and u^h to the primal subgrid model* (4.14)*, we have the error representation*

$$
\hat{\mathcal{E}}_1 = J(u) - J(u^h) = {}_{\mathcal{V}^*}\big\langle q, \tau^e \mathcal{R}u^h\big\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}^*}\langle q, \widetilde{u}\rangle_{\mathcal{V}}\,. \tag{4.37}
$$

Proof.

$$\begin{aligned}
J(u) - J(u^h) &= {}_{\mathcal{V}^*}\langle q, u\rangle_{\mathcal{V}} - {}_{\mathcal{V}^*}\langle q, u^h\rangle_{\mathcal{V}} && \text{by (4.21)}\\
&= {}_{\mathcal{V}^*}\langle q, u - u^h\rangle_{\mathcal{V}} && \text{by linearity}\\
&= {}_{\mathcal{V}^*}\langle q, \widetilde{u}' + \widetilde{u}\rangle_{\mathcal{V}} && \text{by (4.19)}\\
&= {}_{\mathcal{V}^*}\langle q, \widetilde{u}'\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}^*}\langle q, \widetilde{u}\rangle_{\mathcal{V}} && \text{by linearity}\\
&= {}_{\mathcal{V}^*}\langle q, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}^*}\langle q, \widetilde{u}\rangle_{\mathcal{V}}\,. && \text{by (4.16)}
\end{aligned}$$

□

Proposition 4. *For any solutions u to the primal model* (4.2)*, z to the dual model* (4.22)*, u^h to the primal subgrid model* (4.14) *and z^h to the dual subgrid model* (4.30)*, we have the error representation*

$$\begin{aligned}
\hat{\mathcal{E}}_2 &= J(u) - J(u^h)\\
&= {}_{\mathcal{V}}\langle \tau_d^e\mathcal{R}^* z^h, \mathcal{R}u^h\rangle_{\mathcal{V}^*}^{\Omega'} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* z^h, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}}\langle \widetilde{z}, \mathcal{R}u^h\rangle_{\mathcal{V}^*}\,.
\end{aligned} \tag{4.38}$$

Proof.

$$\begin{aligned}
&J(u) - J(u^h)\\
&= {}_{\mathcal{V}^*}\langle q, u\rangle_{\mathcal{V}} - {}_{\mathcal{V}^*}\langle q, u^h\rangle_{\mathcal{V}} && \text{by (4.21)}\\
&= {}_{\mathcal{V}^*}\langle \mathcal{L}^* z, u\rangle_{\mathcal{V}} - {}_{\mathcal{V}^*}\langle \mathcal{L}^* z, u^h\rangle_{\mathcal{V}} && \text{by (4.22)}\\
&= {}_{\mathcal{V}}\langle z, \mathcal{L}u\rangle_{\mathcal{V}^*} - {}_{\mathcal{V}}\langle z, \mathcal{L}u^h\rangle_{\mathcal{V}^*} && \text{by (4.9)}\\
&= {}_{\mathcal{V}}\langle z, f\rangle_{\mathcal{V}^*} - {}_{\mathcal{V}}\langle z, \mathcal{L}u^h\rangle_{\mathcal{V}^*} && \text{by (4.2)}\\
&= {}_{\mathcal{V}}\langle z, \mathcal{R}u^h\rangle_{\mathcal{V}^*} && \text{by definition}\\
&= {}_{\mathcal{V}}\langle z, \mathcal{R}u^h\rangle_{\mathcal{V}^*} - {}_{\mathcal{V}}\langle z^h, \mathcal{R}u^h\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* z^h, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} && \text{by (4.15)}\\
&= {}_{\mathcal{V}}\langle z - z^h, \mathcal{R}u^h\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* z^h, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} && \text{by linearity}\\
&= {}_{\mathcal{V}}\langle \widetilde{z}' + \widetilde{z}, \mathcal{R}u^h\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* z^h, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} && \text{by (4.33)}\\
&= {}_{\mathcal{V}}\langle \widetilde{z}', \mathcal{R}u^h\rangle_{\mathcal{V}^*}^{\Omega'} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* z^h, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}}\langle \widetilde{z}, \mathcal{R}u^h\rangle_{\mathcal{V}^*} && \text{by linearity}\\
&= {}_{\mathcal{V}}\langle \tau_d^e\mathcal{R}^* z^h, \mathcal{R}u^h\rangle_{\mathcal{V}^*}^{\Omega'} + {}_{\mathcal{V}^*}\langle \mathcal{L}^* z^h, \tau^e\mathcal{R}u^h\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}}\langle \widetilde{z}, \mathcal{R}u^h\rangle_{\mathcal{V}^*}\,. && \text{by (4.32)}
\end{aligned}$$

□

4.4.3 Subgrid Model Error Estimates

In general, the approximation errors $\widetilde{u}$ and $\widetilde{z}$ are unknown. This suggests the error *estimates* $\eta_1 \approx \hat{\mathcal{E}}_1$ and $\eta_2 \approx \hat{\mathcal{E}}_2$ that are obtained by setting $\widetilde{u} = 0$ in (4.37) and $\widetilde{z} = 0$ in (4.38), and are given below:

$$\eta_1 = {}_{\mathcal{V}^*}\big\langle q, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'}, \tag{4.39}$$

$$\eta_2 = {}_{\mathcal{V}}\big\langle \tau_d^e \mathcal{R}^* z^h, \mathcal{R} u^h \big\rangle_{\mathcal{V}^*}^{\Omega'} + {}_{\mathcal{V}^*}\big\langle \mathcal{L}^* z^h, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'}. \tag{4.40}$$

Proposition 5. *For any solutions u^h to the primal subgrid model* (4.14) *and z^h to the dual subgrid model* (4.30) *the error estimates η_1 and η_2 are identical.*

Proof. Note that if the stabilization parameters τ^e and τ_d^e for the primal and dual problems are equal, we obtain the desired result since

$$\begin{aligned}
\eta_2 &= {}_{\mathcal{V}}\big\langle \tau_d^e \mathcal{R}^* z^h, \mathcal{R} u^h \big\rangle_{\mathcal{V}^*}^{\Omega'} + {}_{\mathcal{V}^*}\big\langle \mathcal{L}^* z^h, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'} \\
&= {}_{\mathcal{V}^*}\big\langle \mathcal{R}^* z^h, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'} + {}_{\mathcal{V}^*}\big\langle \mathcal{L}^* z^h, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'} && \text{by assumption} \\
&= {}_{\mathcal{V}^*}\big\langle \mathcal{R}^* z^h + \mathcal{L}^* z^h, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'} && \text{by linearity} \\
&= {}_{\mathcal{V}^*}\big\langle q, \tau^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega'} && \text{by definition} \\
&= \eta_1.
\end{aligned}$$

Using the given definitions (4.13) and (4.29), we note that a sufficient condition for the equality $\tau^e = \tau_d^e$ is: $g^e(\boldsymbol{x}; \boldsymbol{y}) = g_d^e(\boldsymbol{y}; \boldsymbol{x})$. This is verified via the following argument:

$$\begin{aligned}
&\mathcal{L}^* g^e(\boldsymbol{x}; \boldsymbol{y}) = \delta(\boldsymbol{x} - \boldsymbol{y}) && \text{by (4.12)} \\
\implies &\int_{\Omega^e} g_d^e(\boldsymbol{x}; \boldsymbol{z}) \mathcal{L}^* g^e(\boldsymbol{x}; \boldsymbol{y}) \, \mathrm{d}\Omega = \int_{\Omega^e} g_d^e(\boldsymbol{x}; \boldsymbol{z}) \delta(\boldsymbol{x} - \boldsymbol{y}) \, \mathrm{d}\Omega \\
\implies &\int_{\Omega^e} \mathcal{L} g_d^e(\boldsymbol{x}; \boldsymbol{z}) g^e(\boldsymbol{x}; \boldsymbol{y}) \, \mathrm{d}\Omega = \int_{\Omega^e} g_d^e(\boldsymbol{x}; \boldsymbol{z}) \delta(\boldsymbol{x} - \boldsymbol{y}) \, \mathrm{d}\Omega \\
\implies &\int_{\Omega^e} \delta(\boldsymbol{x} - \boldsymbol{z}) g^e(\boldsymbol{x}; \boldsymbol{y}) \, \mathrm{d}\Omega = \int_{\Omega^e} g_d^e(\boldsymbol{x}; \boldsymbol{z}) \delta(\boldsymbol{x} - \boldsymbol{y}) \, \mathrm{d}\Omega && \text{by (4.28)} \\
&\implies g^e(\boldsymbol{z}; \boldsymbol{y}) = g_d^e(\boldsymbol{y}; \boldsymbol{z}).
\end{aligned}$$

Here we remark that the identity (4.9) holds for arbitrary smooth domains Ω and for a

function space $\mathcal{V}$ whose members vanish on the boundary $\partial\Omega$. As such, we employ the element-level identity:

$$ {}_{\mathcal{V}}\langle v, \mathcal{L}u\rangle_{\mathcal{V}^*}^{\Omega^e} = {}_{\mathcal{V}^*}\langle \mathcal{L}^* v, u\rangle_{\mathcal{V}}^{\Omega^e} \quad \forall u, v \in \mathcal{V}^e \tag{4.41} $$

to derive the third equality above, where $\mathcal{V}^e = \{u \in \mathcal{V} : u = 0 \text{ on } \partial\Omega^e\}$. □

4.4.4 Error Localization

We now demonstrate that even though η_1 and η_2 are identical global error estimates, their localization to element-level error estimates is very different. This localization yields positive values at the element level called *error indicators* which are necessary to drive mesh adaptation. We compute error indicators by bounding the two error estimates η_1 and η_2 from above using the triangle inequality, such that:

$$ |\eta_1| \leq \sum_{e=1}^{n_{el}} \eta_1^e, \tag{4.42} $$

and

$$ |\eta_2| \leq \sum_{e=1}^{n_{el}} \eta_2^e. \tag{4.43} $$

Here the error indicator for the error estimate η_1 is given as

$$ \eta_1^e = |\, {}_{\mathcal{V}^*}\big\langle q, \boldsymbol{\tau}^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega^e} \,|, \tag{4.44} $$

and the error indicator for the error estimate η_2 is given as

$$ \eta_2^e = |\, {}_{\mathcal{V}}\big\langle \boldsymbol{\tau}_d^e \mathcal{R}^* z^h, \mathcal{R} u^h \big\rangle_{\mathcal{V}^*}^{\Omega^e} \,| + |\, {}_{\mathcal{V}^*}\big\langle \mathcal{L}^* z^h, \boldsymbol{\tau}^e \mathcal{R} u^h \big\rangle_{\mathcal{V}}^{\Omega^e} \,|. \tag{4.45} $$

Note that the indicator η_1^e is only non-zero over elements for which the dual forcing function $q|_{\Omega^e}$ is non-zero. This indicates that only elements for which $q|_\Omega \neq 0$ provide contributions to the error $J(u) - J(u^h)$, which is generally not true. As a thought experiment, consider an advective problem for which the dual forcing function q is defined to be 1 over some subdomain $\Omega^s \subset \Omega$ and 0 elsewhere. Any discretization errors introduced upstream of

the subdomain Ω^s will be propagated via advection to the subdomain itself, thus affecting the accuracy of the computed output quantity. However, the indicator η_1^e will indicate that the elements upstream of the subdomain provide no contributions to the output error, as these elements are located outside of the subdomain Ω^s, whereas this would not be the case for η_2^e.

4.5 Mesh Adaptation

Mesh adaptation provides a means to modify the spatial discretization of a PDE to obtain greater solution accuracy with a given amount of computing power. Presently, we make use of conformal unstructured local mesh modification that performs sequences of edge splits, swaps, and collapses [18] [17] using the PUMI [10] software suite. Mesh adaptation is driven by the concept of a *mesh size field*, which defines element edge lengths at all locations in the mesh. The mesh size field is determined by the localized error indicators to perform mesh refinement in areas that strongly contribute to the error and perform mesh coarsening in areas that do not strongly contribute to the error.

4.5.1 Size Field Specification

Let N be a desired target number of mesh elements. Let η^e denote a computed element-level error indicator defined for all $e = 1, 2, \ldots, n_{el}$. Let p be the expected polynomial order of convergence for a chosen finite element method. Following Boussetta et al. [34], we utilize a size field specification that aims to provide an output adapted mesh with N elements. First, we define the global quantity G as

$$G = \sum_{e}^{n_{el}} (\eta^e)^{\frac{2d}{2p+d}} . \tag{4.46}$$

Once G has been computed, new element-level sizes h^e_{new} are determined by scaling the previous element size h_e according to the formula

$$h^e_{\text{new}} = \left(\frac{G}{N}\right)^{\frac{1}{d}} (\eta^e)^{\frac{-2}{2p+d}} \, h^e. \tag{4.47}$$

Finally, to prevent excessive refinement or coarsening in a single adaptive step, we prescribe that the new element size be no smaller than half the previous element size and no greater

than twice the previous element size.

$$\frac{1}{2} \leq \frac{h^e_{\text{new}}}{h^e} \leq 2. \tag{4.48}$$

4.6 Results

In this section, we investigate output-based error estimation and mesh adaptation as applied to a model scalar, steady state advection diffusion problem, defined by the linear operator

$$\mathcal{L} := -\kappa\nabla^2 + \boldsymbol{a} \cdot \nabla. \tag{4.49}$$

Here, κ is a coefficient corresponding to the diffusivity strength and $\boldsymbol{a}$ is a coefficient corresponding to the advective transport. The adjoint operator is readily found (see B.2) to be

$$\mathcal{L}^* = -\kappa\nabla^2 - \boldsymbol{a} \cdot \nabla, \tag{4.50}$$

which is simply another advection-diffusion operator with the advective direction opposite that of the original operator. The bilinear form $A(\cdot, \cdot)$ associated with the operator $\mathcal{L}$ is given as

$$A(v, u) = (\nabla v, \kappa\nabla u) + (v, \boldsymbol{a} \cdot \nabla u) \tag{4.51}$$

where $(\cdot, \cdot)$ denotes the L_2 inner product.

The mesh Peclét number α is given by $\alpha := \frac{h|\boldsymbol{a}|}{2\kappa}$, where $h = \text{meas}(\Omega^e)$ is a characteristic measure of the mesh element size. In one dimension, the stabilization parameter $\boldsymbol{\tau}^e$ is given [55] as:

$$\boldsymbol{\tau}^e = \frac{h}{2|\boldsymbol{a}|}(\coth\alpha + \frac{1}{\alpha}). \tag{4.52}$$

The parameter $\boldsymbol{\tau}^e$ exactly solves (4.13) in one spatial dimension, but we emphasize that utilizing this parameter in two spatial dimensions introduces yet another approximation to the fine-scale solution.

For a chosen functional output quantity $J(u)$, the *effectivity index* is defined as

$$I = \frac{J(u) - J(u^h)}{\eta}, \tag{4.53}$$

the ratio of the exact error to the estimated error. The effectivity index provides a measure of the degree to which the error is underestimated. An effectivity index of $I = 1$ is desirable, as it indicates the error estimate has exactly recovered the error.

For each numerical example, the primal and dual problems are solved using the same finite element discretization. That is the same finite element basis functions and the same finite element mesh are used to solve the primal and dual problems. The mesh used in each example consists of simplical elements in one or two dimensions, and the finite element subspace $\mathcal{V}^h$ is defined by piecewise linear Lagrange shape functions. The primal problem is given by equation (4.20) and the dual problem is given by equation (4.34), where we emphasize that homogeneous Dirichlet boundary conditions are applied to both the primal and dual problems. Finally, we note that we have provided B.3 to concretely demonstrate the propositions derived in Section 4.4.

4.6.1 One Dimensional Example

Let $\Omega = \{x : x \in [0, 1]\}$. We choose the forcing function for the primal to be $f = 1$, and the functional quantity of interest $J(u) = \int_\Omega u \, \mathrm{d}\Omega$, such that $q = 1$ for the dual problem. The diffusivity coefficient is chosen to be $\kappa = 0.001$ and the advective coefficient $a = 1$. The exact solution to the primal PDE (4.1) is

$$u(x) = \frac{1}{a}\left(x - \frac{\exp(\frac{ax}{\kappa}) - 1}{\exp(\frac{a}{\kappa}) - 1}\right) \tag{4.54}$$

and the exact value for the chosen quantity of interest is $J(u) = 0.499$.

We investigate the accuracy of the error estimate obtained by $\eta = \eta_1 = \eta_2$. Table 4.1 shows the computed functional quantity of interest and the effectivity indices for the one-dimensional problem solved on meshes with n_{el} elements with mesh size $h = \frac{1}{n_{el}}$. For each chosen mesh size, the effectivity index is exactly one meaning the error estimate η exactly recovers the output error. It is well known (*c.f.* [55]) that our choice of $\boldsymbol{\tau}^e$ results in a solution u^h that is nodally exact. For this reason, it is unsurprising that the output error is exactly recovered for this example.

Table 4.1. Effectivity indices for a 1D advection-diffusion example with a global QoI.

n_{el}	α	$J(u^h)$	I
10	5.000e+01	4.5000e-01	1.000
20	2.500e+01	4.7500e-01	1.000
40	1.250e+01	4.8750e-01	1.000
80	6.250e+00	4.9375e-01	1.000
160	3.125e+00	4.9686e-01	1.000

4.6.2 A Manufactured Solution

Let $\Omega = \{\boldsymbol{x} : \boldsymbol{x} \in [0,1] \times [0,1]\}$. Let $\boldsymbol{e_i}$ and $\boldsymbol{e_j}$ be unitary vectors in the x and y directions, respectively. We choose the advective coefficient to be $\boldsymbol{a} = \boldsymbol{e_i} + \boldsymbol{e_j}$, the diffusive coefficient to be $\kappa = 0.001$, and the forcing function f such that the exact solution is given by

$$u(x, y) = \sin(\pi x) \sin(\pi y). \tag{4.55}$$

The quantity of interest is chosen to be $J(u) = \int_\Omega u \, \mathrm{d}\Omega$, such that the dual forcing function is $q = 1$. The exact value of the quantity of interest is $J(u) = \frac{4}{\pi^2} \approx .405284$. Again, we investigate the effectivity of the error estimate $\eta = \eta_1 = \eta_2$ for meshes with uniformly n_{el} uniformly distributed triangular elements. Table 4.2 shows effectivity indices obtained for various meshes. As the mesh size decreases and the number of elements increases, the effectivity index tends to one.

Table 4.2. Effectivity indices for a 2D advection-diffusion example with a global QoI.

n_{el}	$J(u^h)$	I
200	4.0493e-01	1.083
800	4.0512e-01	1.023
3200	4.0521e-01	1.009
12800	4.0525e-01	1.004
51200	4.0527e-01	1.001

4.6.3 Advection in an L-Shaped Domain

Let $\Omega = \{\boldsymbol{x} : \boldsymbol{x} \in [0,1] \times [0,1] \cup [0,1] \times [-1,0] \cup [-1,0] \times [0,1]\}$. Let $\boldsymbol{e_i}$ and $\boldsymbol{e_j}$ be unitary vectors in the x and y direction, respectively. We choose the advective coefficient to be $\boldsymbol{a} = -\boldsymbol{e_i} + \boldsymbol{e_j}$, the diffusive coefficient to be $\kappa = 0.001$, and the forcing function $f = 1$. We investigate adaptivity for two output quantities: $J_1(u) = \int_\Omega u \, \mathrm{d}\Omega$ and $J_2(u) = \int_\Omega q_2 u \, \mathrm{d}\Omega$, where q_2 is defined as

$$q_2 := \begin{cases} 1 & \text{if } -0.95 \leq x \leq -0.5 \text{ and } 0.5 \leq y \leq 0.95 \\ 0 & \text{otherwise.} \end{cases} \tag{4.56}$$

That is, q_2 samples the solution u^h on a square patch in the upper right corner of the domain Ω. The primal solution u^h and the dual solutions corresponding to the two quantities of interest are shown in Figure 4.1. Note that the primal solution contains steep gradients at the two left-most surfaces and the upper surface of the L-shaped domain.

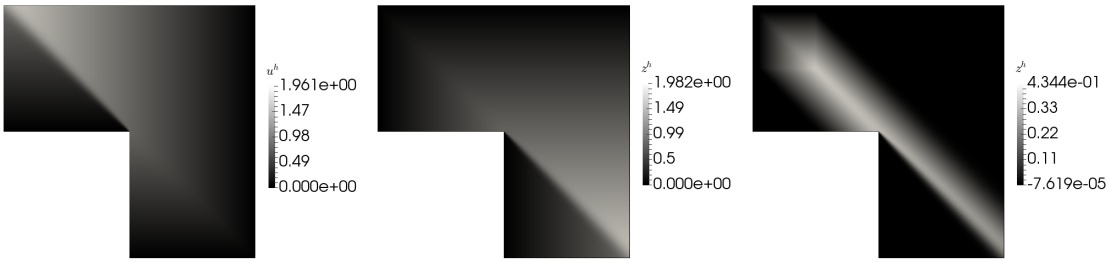

Fig. 4.1. The primal solution u^h (left) and the dual solutions z^h corresponding to $J_1(u)$ (center) and $J_2(u)$ (right).

We investigate the ability of four adaptive schemes to accurately assess the two functional quantities. Each scheme proceeds by iteratively performing the steps

$$\text{Solve primal PDE} \rightarrow \text{Localize error} \rightarrow \text{Adapt mesh.}$$

The first adaptive scheme, referred to as UNIF, remeshes the entire domain with a uniform size field. For the two output quantities, errors are computed for the meshes generated with the mesh sizes $h = \{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$. In principle, the step to localize the error is not required for this scheme.

For comparison to more traditional energy-based methods, the second adaptive scheme

is chosen based on a Zienkiewicz-Zhu type error estimate [67] [68], whereby error indicators are computed as the difference between solution gradients ∇u^h that are discontinuous between elements and a nodally smoothed approximation to the gradient $(\nabla u^h)^*$ that is obtained via a least-squares fit over a patch of elements. Once error indicators are computed, the size field is set according to the size field equation (4.47) such that the target number of elements N is twice the number of elements in the previous mesh. We refer to this scheme as the superconvergent patch recovery (SPR) adaptive scheme.

The third and fourth adaptive schemes are based on the error indicators η_1^e and η_2^e, respectively, and are referred to as the VMS1 and VMS2 adaptive schemes, respectively. Again, once the error indicators have been computed, the size field is set according to the size field equation (4.47) such that the target number of elements N is twice the number of elements in the previous mesh. We note that the scheme VMS2 is the only one which necessitates the solution of the dual PDE model, which is implicitly included in the 'Localize error' step.

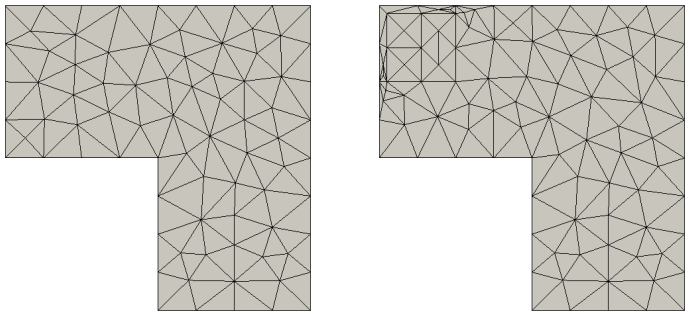

Fig. 4.2. Initial meshes for the outputs $J_1(u)$ (left) and $J_2(u)$ (right).

For each quantity of interest, an initial mesh with a uniform size of $h = \frac{1}{4}$ was generated as shown in Figure 4.2. From this initial mesh, each adaptive scheme was run until meshes with over $10,000$ degrees of freedom were produced. The exact values of the two output quantities were computed on 'truth' meshes, which are finer at every spatial location in the domain when compared to the meshes obtained via the four adaptive schemes. The values of the quantities of interest were found to be $J_1(u) = 1.6588688371$ and $J_2(u) = 0.23109653499$.

Figure 4.3 shows the meshes obtained at the final iteration of the SPR, VMS1, and VMS2 adaptive schemes for the global output quantity $J_1(u)$. As expected, the SPR scheme strongly refines the mesh in areas where the gradient changes drastically. These areas include

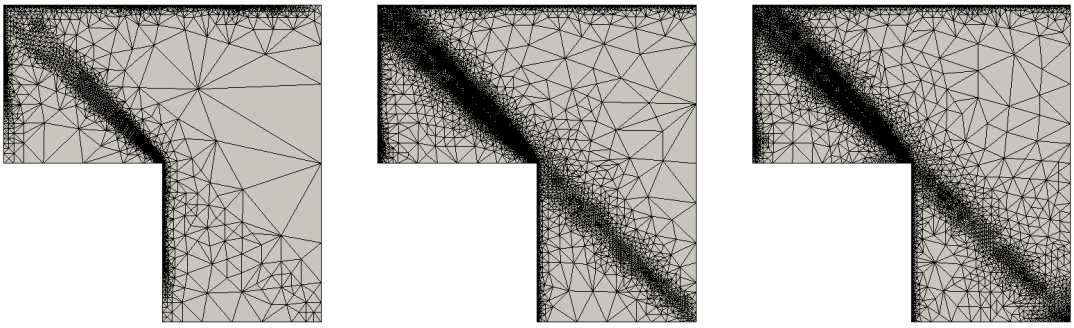

Fig. 4.3. Final adapted meshes for the output $J_1(u)$ using the SPR (left), VMS1 (center), and VMS2 (right) adaptive schemes.

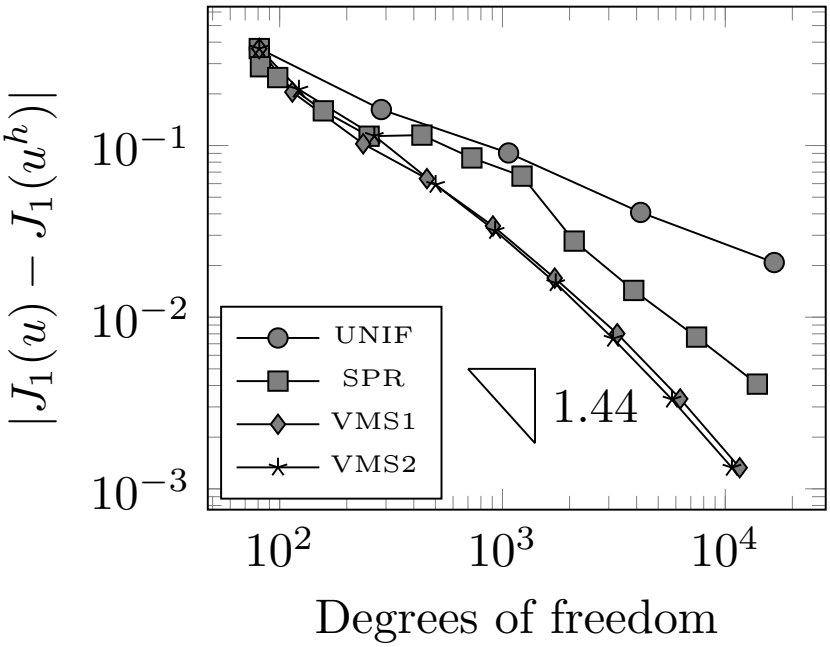

Fig. 4.4. Convergence history for various adaptive schemes for the output $J_1(u)$.

the left-most and upper-most surfaces of the L-shaped domain where boundary layers in the solution exist, as well as the diagonal downstream of the reentrant corner where there is a sudden change in the solution magnitude. In addition to performing mesh refinement in the areas that the SPR scheme targets, the VMS1 and VMS2 also refine the mesh along the diagonal upstream of the reentrant corner to accurately resolve features of the dual solution z^h. For the global quantity $J_1(u)$, the VMS1 and VMS2 schemes yield final

meshes with very similar characteristics. At each iteration in the adaptive schemes, the output error $|J_1(u) - J_1(u^h)|$ was computed. Figure 4.4 displays the convergence histories for each adaptive scheme. Unsurprisingly, the VMS1 and VMS2 adaptive schemes compute the output error more accurately than the SPR and UNIF with a comparable number of degrees of freedom.

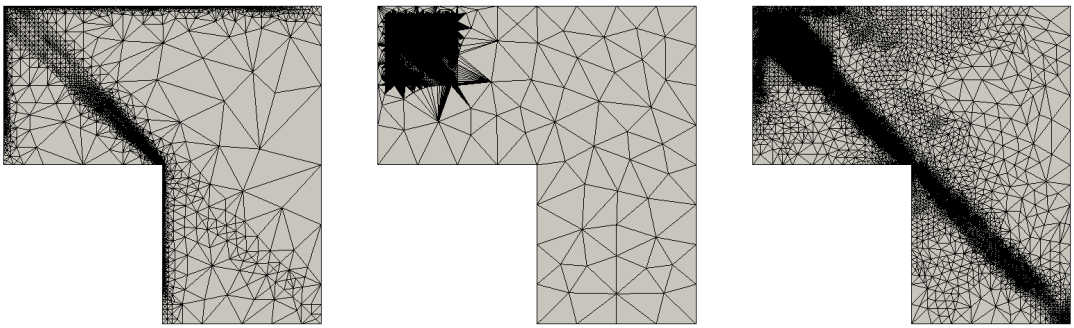

Fig. 4.5. Final adapted meshes for the output $J_2(u)$ using the SPR (left), VMS1 (center), and VMS2 (right) adaptive schemes.

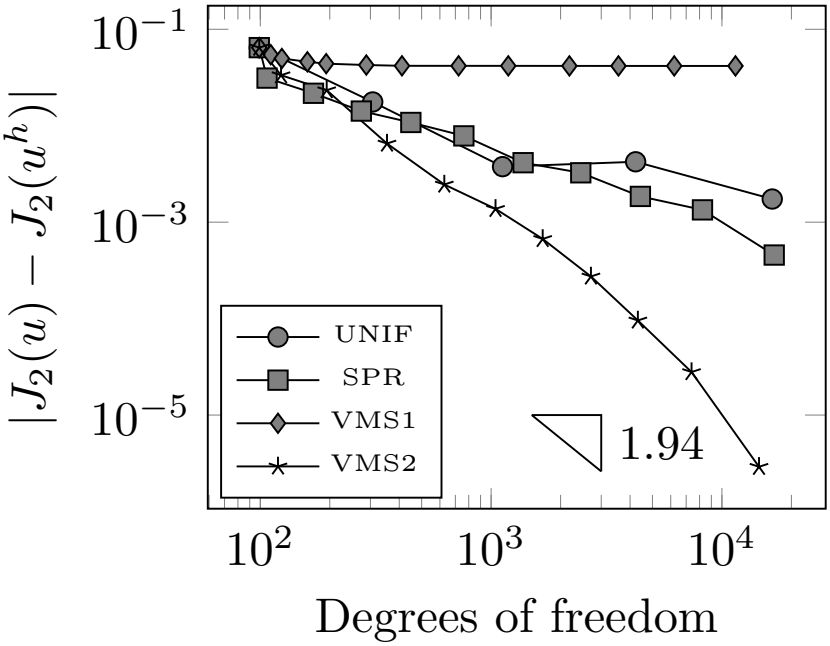

Fig. 4.6. Convergence history for various adaptive schemes for the output $J_2(u)$.

Figure 4.5 displays the output meshes at the final iteration of the SPR, VMS1, and

VMS2 adaptive schemes for the local output quantity $J_2(u)$. Again, it is clear that the SPR strongly refines the mesh in areas where the gradient changes drastically. The VMS1 scheme only performs mesh refinement over the square subdomain over which q_2 is non-zero and does not seek accurately resolve the mesh to capture features of the primal or dual solutions. In contrast, the VMS2 strongly refines the mesh over the square subdomain of interest, while also resolving areas upstream of the domain to accurately account for the features of the primal and dual solutions. For the output quantity $J_2(u)$, convergence histories for each adaptive scheme are shown in Figure 4.6. It is clear from both the convergence diagram and final adapted mesh for the VMS1 scheme that the VMS1 scheme is completely insufficient to drive mesh adaptation for a locally defined output quantity. On the other hand, the VMS2 adaptive scheme is able to compute the ouptut quantity with much greater accuracy than the UNIF and SPR schemes when using a comparable number of degrees of freedom.

4.7 Conclusions

For VMS methods, we have proposed a novel approach to enriching the dual solution for duality-based functional error estimation using VMS techniques. We have demonstrated the utility of this technique to drive mesh adaptation to accurately compute output quantities.

Future work includes investigating the effect of choosing different optimality conditions $\phi(\cdot)$ and $\phi_d(\cdot)$ for the primal and dual problems, respectively, extending error estimates to account for nonlinearities in both the PDE model and in the functional output quantity, investigating the effect of utilizing more accurate approximations to the fine-scale primal and dual solutions, and extending the arguments presented to a mixture of non-homogeneous Dirichlet and Nuemann boundary conditions.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

REFERENCES

[1] M. E. Rognes and A. Logg, "Automated goal-oriented error control I: Stationary variational problems," *SIAM J. on Scientific Comput.*, vol. 35, no. 3, pp. C173–C193, May. 2013.

[2] A. Logg, K.-A. Mardal, and G. Wells, *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Heidelberg, Germany: Springer, 2012.

[3] T. Richter and T. Wick, "Variational localizations of the dual weighted residual estimator," *J. of Comput. and Appl. Math.*, vol. 279, pp. 192–208, May. 2015.

[4] B. N. Granzow. *Goal GitHub Repository*. (2017) [Online]. Available: https://github.com/bgranzow/goal, Accessed on: Dec. 1, 2017.

[5] R. P. Pawlowski, E. T. Phipps, and A. G. Salinger, "Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part I: Template-based generic programming," *Scientific Programming*, vol. 20, no. 2, pp. 197–219, Apr. 2012.

[6] R. P. Pawlowski, E. T. Phipps, A. G. Salinger, S. J. Owen, C. M. Siefert, and M. L. Staten, "Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part II: Application to partial differential equations," *Scientific Programming*, vol. 20, no. 3, pp. 327–345, Jul. 2012.

[7] R. Becker and R. Rannacher, "An optimal control approach to a posteriori error estimation in finite element methods," *Acta Numerica*, vol. 10, pp. 1–102, May. 2001.

[8] E. C. Cyr, J. Shadid, and T. Wildey, "Approaches for adjoint-based a posteriori analysis of stabilized finite element methods," *SIAM J. on Scientific Comput.*, vol. 36, no. 2, pp. A766–A791, Apr. 2014.

[9] K. J. Fidkowski and D. L. Darmofal, "Review of output-based error estimation and mesh adaptation in computational fluid dynamics," *AIAA J.*, vol. 49, no. 4, pp. 673–694, Apr. 2011.

[10] D. A. Ibanez, E. S. Seol, C. W. Smith, and M. S. Shephard, "PUMI: Parallel unstructured mesh infrastructure," *ACM Trans. on Math. Software*, vol. 42, no. 3, pp. 17–45, Jun. 2016.

[11] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda *et al.*, "An overview of the Trilinos project," *ACM Trans. on Math. Software*, vol. 31, no. 3, pp. 397–423, Sep. 2005.

[12] M. A. Heroux and J. M. Willenbring, "A new overview of the Trilinos project," *Scientific Programming*, vol. 20, no. 2, pp. 83–88, Mar. 2012.

[13] E. Phipps and R. Pawlowski, "Efficient expression templates for operator overloading-based automatic differentiation," in *Recent Advances in Algorithmic Differentiation*. Berlin, Germany: Springer, 2012, pp. 309–319.

[14] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, "Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems," *Scientific Programming*, vol. 20, no. 3, pp. 241–255, Jan. 2012.

[15] A. Prokopenko, J. J. Hu, T. A. Wiesner, C. M. Siefert, and R. S. Tuminaro, "MueLu users guide 1.0," Sandia Nat. Lab., Albuquerque, NM, USA, Tech. Rep. SAND2014-18874, Oct. 2014.

[16] D. Ibanez and M. S. Shephard, "Modifiable array data structures for mesh topology," *SIAM J. on Scientific Comput.*, vol. 39, no. 2, pp. C144–C161, Apr. 2017.

[17] X. Li, M. S. Shephard, and M. W. Beall, "3D anisotropic mesh adaptation by mesh modification," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 194, no. 48, pp. 4915–4950, Nov. 2005.

[18] F. Alauzet, X. Li, E. S. Seol, and M. S. Shephard, "Parallel anisotropic 3D mesh adaptation by mesh modification," *Eng. with Comp.*, vol. 21, no. 3, pp. 247–258, Jan. 2006.

[19] C. W. Smith, M. Rasquin, D. Ibanez, K. E. Jansen, and M. S. Shephard, "Improving unstructured mesh partitions for multiple criteria using mesh adjacencies," *SIAM J. Scientific Comput.*, to be published.

[20] G. Diamond, C. W. Smith, and M. S. Shephard, "Dynamic load balancing of massively parallel unstructured meshes," in *Proc. of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, Denver, CO, USA*, Denver, CO, USA, Nov. 2017.

[21] C. W. Smith, B. Granzow, D. Ibanez, O. Sahni, K. E. Jansen, and M. S. Shephard, "In-memory integration of existing software components for parallel adaptive unstructured mesh workflows," in *Proc. of the XSEDE16 Conf. on Diversity, Big Data, and Science at Scale, Miami, FL, USA*, Miami, FL, USA, Jul. 2016.

[22] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed. Philadelphia, PA, USA: Soc. for Ind. & Appl. Math., 2008.

[23] A. G. Salinger, R. A. Bartett, Q. Chen, X. Gao, G. Hansen, I. Kalashnikova *et al.*, "Albany: A component-based partial differential equation code built on trilinos." Sandia Nat. Lab., Albuquerque, NM, USA, Tech. Rep. SAND2013-8430J, Nov. 2013.

[24] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price, "Albany/FELIX: A parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis," *Geoscientific Model Development*, vol. 8, no. 4, pp. 1197–1220, Apr. 2015.

[25] B. Ramesh and A. M. Maniatty, "Stabilized finite element formulation for elastic–plastic finite deformations," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 194, no. 6, pp. 775–800, Feb. 2005.

[26] M. Nemec and M. J. Aftosmis, "Adjoint error estimation and adaptive refinement for embedded-boundary Cartesian meshes," presented at the 18th AIAA Computational Fluid Dynamics Conf., Miami, FL, USA, 2007.

[27] B. N. Granzow, M. S. Shephard, and A. A. Oberai, "Output-based error estimation and mesh adaptation for variational multiscale methods," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 322, pp. 441–459, Aug. 2017.

[28] K. J. Fidkowski, "Output error estimation strategies for discontinuous galerkin discretizations of unsteady convection-dominated flows," *Int. J. for Numerical Methods in Eng.*, vol. 88, no. 12, pp. 1297–1322, May. 2011.

[29] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "Parallel scalable adjoint-based adaptive solution of variable-viscosity stokes flow problems," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 198, no. 21, pp. 1691–1700, May. 2009.

[30] C. Taylor and P. Hood, "A numerical solution of the Navier-Stokes equations using the finite element technique," *Comput. & Fluids*, vol. 1, no. 1, pp. 73–100, Jan. 1973.

[31] D. A. Venditti and D. L. Darmofal, "Adjoint error estimation and grid adaptation for functional outputs: Application to quasi-one-dimensional flow," *J. of Comput. Physics*, vol. 164, no. 1, pp. 204–227, Oct. 2000.

[32] ——, "Grid adaptation for functional outputs: Application to two-dimensional inviscid flows," *J. of Comput. Physics*, vol. 176, no. 1, pp. 40–69, Feb. 2002.

[33] ——, "Anisotropic grid adaptation for functional outputs: Application to two-dimensional viscous flows," *J. Comput. Phys.*, vol. 187, no. 1, pp. 22–46, May. 2003.

[34] R. Boussetta, T. Coupez, and L. Fourment, "Adaptive remeshing based on a posteriori error estimation for forging simulation," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 195, no. 48, pp. 6626–6645, Oct. 2006.

[35] W. Bangerth. *Deal ii Step 14*. (2017) [Online]. Available: https://www.dealii.org/8.4.0/doxygen/deal.II/step_14.html, Accessed on: Dec. 1, 2017.

[36] B. N. Granzow, A. A. Oberai, and M. S. Shephard, "Adjoint-based error estimation and mesh adaptation for stabilized finite deformation elasticity," submitted for publication.

[37] L. Dong and A. A. Oberai, "Recovery of cellular traction in three-dimensional nonlinear hyperelastic matrices," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 314, pp. 296–313, Feb. 2017.

[38] M. O. Bloomfield, Z. Li, B. Granzow, D. A. Ibanez, A. A. Oberai, G. A. Hansen *et al.*, "Component-based workflows for parallel thermomechanical analysis of arrayed geometries," *Eng. with Comput.*, vol. 33, no. 3, pp. 509–517, Jul. 2017.

[39] J. C. Simo and T. J. Hughes, *Computational Inelasticity*. New York, NY, USA: Springer, 2006.

[40] Z. Li, M. O. Bloomfield, and A. A. Oberai, "Simulation of finite-strain inelastic phenomena governed by creep and plasticity," *Comput. Mechanics*, to be published.

[41] L. F. Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philosophical Trans. of the Royal Society of London*, vol. 210, pp. 307–357, Jan. 1911.

[42] M. B. Giles and N. A. Pierce, "Chapter 2 - Adjoint error correction for integral outputs," in *Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics*. Berlin, Germany: Springer, 2016, pp. 47–95.

[43] N. A. Pierce and M. B. Giles, "Adjoint and defect error bounding and correction for functional estimates," *J. of Comput. Physics*, vol. 200, no. 2, pp. 769–794, Nov. 2004.

[44] S. Prudhomme and J. T. Oden, "On goal-oriented error estimation for elliptic problems: Application to the control of pointwise errors," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 176, no. 1-4, pp. 313–331, Jul. 1999.

[45] S. Prudhomme, J. T. Oden, T. Westermann, J. Bass, and M. E. Botkin, "Practical methods for a posteriori error estimation in engineering applications," *Int. J. for Numerical Methods. in Eng.*, vol. 56, no. 8, pp. 1193–1224, Jan. 2003.

[46] J. M. Connors, J. W. Banks, J. A. Hittinger, and C. S. Woodward, "A method to calculate numerical errors using adjoint error estimation for linear advection," *SIAM J. on Numerical Anal.*, vol. 51, no. 2, pp. 894–926, Mar. 2013.

[47] T. Wick, "Goal functional evaluations for phase-field fracture using PU-based DWR mesh adaptivity," *Comput. Mechanics*, vol. 57, no. 6, pp. 1017–1035, Mar. 2016.

[48] L. P. Franca, S. L. Frey, and T. J. Hughes, "Stabilized finite element methods: I. Application to the advective-diffusive model," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 95, no. 2, pp. 253–276, Mar. 1992.

[49] T. J. Hughes, L. P. Franca, and G. M. Hulbert, "A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-diffusive equations," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 73, no. 2, pp. 173–189, May. 1989.

[50] T. J. Hughes, L. P. Franca, and M. Balestra, "A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška-Brezzi condition: A stable Petrov-Galerkin formulation of the Stokes problem accommodating equal-order interpolations," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 59, no. 1, pp. 85–99, Nov. 1986.

[51] T. Barth, P. Bochev, M. Gunzburger, and J. Shadid, "A taxonomy of consistently stabilized finite element methods for the Stokes problem," *SIAM J. on Scientific Comput.*, vol. 25, no. 5, pp. 1585–1607, Jul. 2004.

[52] A. N. Brooks and T. J. Hughes, "Streamline upwind Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 32, no. 1, pp. 199–259, Sep. 1982.

[53] L. P. Franca and S. L. Frey, "Stabilized finite element methods: II. The incompressible Navier-Stokes equations," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 95, no. 2, pp. 209–233, Sep. 1992.

[54] T. E. Tezduyar, S. Mittal, S. Ray, and R. Shih, "Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 95, no. 2, pp. 221–242, Mar. 1992.

[55] T. J. Hughes, G. R. Feijóo, L. Mazzei, and J.-B. Quincy, "The variational multiscale method: A paradigm for computational mechanics," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 166, no. 1, pp. 3–24, Nov. 1998.

[56] T. J. Hughes and G. Sangalli, "Variational multiscale analysis: The fine-scale Green's function, projection, optimization, localization, and stabilized methods," *SIAM J. on Numerical Anal.*, vol. 45, no. 2, pp. 539–557, Mar. 2007.

[57] M. Ainsworth and J. T. Oden, *A Posteriori Error Estimation in Finite Element Analysis.* Hoboken, NJ, USA: John Wiley & Sons, Ltd, 2011.

[58] G. Hauke, M. H. Doweidar, and M. Miana, "The multiscale approach to error estimation and adaptivity," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 195, no. 13, pp. 1573–1593, Feb. 2006.

[59] G. Hauke, D. Fuster, and M. H. Doweidar, "Variational multiscale a-posteriori error estimation for multi-dimensional transport problems," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 197, no. 33, pp. 2701–2718, Jan. 2008.

[60] A. Masud, T. J. Truster, and L. A. Bergman, "A variational multiscale a posteriori error estimation method for mixed form of nearly incompressible elasticity," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 200, no. 47, pp. 3453–3481, Mar. 2011.

[61] A. Masud and T. J. Truster, "A framework for residual-based stabilization of incompressible finite elasticity: Stabilized formulations and f methods for linear triangles and tetrahedra," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 267, pp. 359–399, Dec. 2013.

[62] M. G. Larson and A. Målqvist, "Adaptive variational multiscale methods based on a posteriori error estimation: energy norm estimates for elliptic problems," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 196, no. 21, pp. 2313–2324, Apr. 2007.

[63] W. Bangerth and R. Rannacher, *Adaptive Finite Element Methods for Differential Equations*. Basel, Switzerland: Birkhäuser Basel, 2013.

[64] G. Hauke and D. Fuster, "Variational multiscale a posteriori error estimation for quantities of interest," *J. of Appl. Mechanics*, vol. 76, no. 2, pp. 21 201–21 207, Jan. 2009.

[65] A. A. Oberai and P. M. Pinsky, "A multiscale finite element method for the Helmholtz equation," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 154, no. 3, pp. 281–297, Mar. 1998.

[66] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson, "Introduction to adaptive methods for differential equations," *Acta Numerica*, vol. 4, pp. 105–158, Jan. 1995.

[67] O. C. Zienkiewicz and J. Z. Zhu, "The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique," *Int. J. for Numerical Methods. in Eng.*, vol. 33, no. 7, pp. 1331–1364, May. 1992.

[68] ——, "The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity," *Int. J. for Numerical Methods. in Eng.*, vol. 33, no. 7, pp. 1365–1382, May. 1992.

APPENDIX A
FORWARD AUTOMATIC DIFFERENTIATION

A.1 Introduction

Automatic differentiation (AD) is a useful technique to computationally evaluate analytic derivatives (to machine precision) of a given function. Numerical software is necessarily executed as a sequence of elementary operations, and AD operates by applying the chain rule to this sequence of elementary operations. There are two standard modes of AD, forward and reverse, as well as two methods of implementing the technique, operator overloading and source code transformation. Presently, we consider the forward mode of automatic differentiation using operator overloading in the context of the `C++` programming language. For an excellent comprehensive overview of automatic differentiation, see [22].

A.2 Forward AD with Operator Overloading

`C++` provides the capability to overload standard operators such as $+, -, *$, and / for custom data types. In scientific computing, operator overloading is convenient in that it allows developers the ability to program using a notation much closer the the target mathematical notation. A common example in scientific computing is the use of operator overloading to implement matrix multiplication. Say we have defined a `C++` dense matrix container called MyMatrix. Using operator overloading, a developer can define matrix multiplication specific behavior for the $*$ operator for the MyMatrix variable. Listing A.1 demonstrates the convenience of using operator overloading to multiply two MyMatrix objects.

Listing A.1. Using operator overloading for a custom matrix class.

```
1 MyMatrix A = ...
2 MyMatrix B = ...
3 MyMatrix C = A*B;
```

In the forward mode AD, a `C++` class variable container is defined that stores both a scalar value x that corresponds the value of the variable and a derivative array $\boldsymbol{x}'$ that corresponds the values of derivatives of the variable with repsect to chosen independent variables. The computer program is then written in terms of these AD variables. At the beginning

Operation	Derivative rule
$c = a \pm b$	$\boldsymbol{c'} = \boldsymbol{a'} \pm \boldsymbol{b'}$
$c = ab$	$\boldsymbol{c'} = a\boldsymbol{b'} + b\boldsymbol{a'}$
$c = a/b$	$\boldsymbol{c'} = (\boldsymbol{a'}b - \boldsymbol{b'}a)/b^2$
$c = \sin(a)$	$\boldsymbol{c'} = \cos(a)\boldsymbol{a'}$
$c = \cos(a)$	$\boldsymbol{c'} = -\sin(a)\boldsymbol{a'}$
$c = \exp(a)$	$\boldsymbol{c'} = \exp(a)\boldsymbol{a'}$
$c = \log(a)$	$\boldsymbol{c'} = \boldsymbol{a'}/a$

of the program, appropriate AD variables are initialized to their appropriate values, and *seeded*. Seeding refers to appropriately setting the derivative array of the AD variables. For example, the i^{th} variable in a system of n independent variables could be represented as:

$$x_i \quad [\frac{\partial x_i}{\partial x_1}, \frac{\partial x_i}{\partial x_2}, \ldots, \frac{\partial x_i}{\partial x_n}], \tag{A.1}$$

where x_i is the variable value and the terms contained in the brackets represent the AD variable's derivative array. To appropriately seed the derivative array, all values then become zero except for the i^{th} component, which becomes 1.

$$x_i \quad [0, 0, \ldots, 1, \ldots, 0, 0]. \tag{A.2}$$

Over the course of the execution of the software program, intermediate derivative arrays are updated via operator overloading when a new operator is encouterd. The operators are overloaded with definitions from basic calclulus rules. For example, the result for the derivative array of an AD variable a times another AD variable b using the overloaded $*$ operator would result in

$$a \cdot b \quad [\boldsymbol{a'} \cdot b + a \cdot \boldsymbol{b'}], \tag{A.3}$$

where the resultant value of the AD variable is simply the multiplied value $a \cdot b$, but the derivative array is updated according to the product rule. Here $\boldsymbol{a'} \cdot b$ is the scalar value of the AD variable b times a's derivative array and $a \cdot \boldsymbol{b'}$ is the scalar value of a times b's derivative array.

Now, if we program the computation of a given function $f(x_i)$, $i = 1, 2, \ldots, n$, with AD variables, we necessarily obtain both the value of the function f and its gradient $\nabla f = \boldsymbol{f'}$.

This is by virtue of the fact that derivatives have been propogated forward through the code from the *seed* point to the evaluation of f.

From an implementation point of view, automatic differentiation via operator overloading is attractive in that existing codes can be easily modified to obtain gradient information. For example, consider a code that uses doubles for all of its evaluation types. A developer could simply replace appropriate instances of 'double' in the code with the appropriate AD type name. Through very little development cost, gradient information is obtained.

A.3 A Simple Example

To further illustrate the concept of forward AD, we present a simple example using Sandia's Sacado automatic differentiation library and examine it in depth. Listing A.2 presents a simple example of the computation of three functions using forward automatic differentiation and displays the resulting values and gradients of the evaluated functions.

Listing A.2. A simple forward AD example.

```
1  #include <Sacado.hpp>
2  typedef Sacado::Fad::DFad<double> FAD;
3  FAD f(FAD x, FAD y) { return x + y; }
4  FAD g(FAD x, FAD y) { return x * y; }
5  int main() {
6    FAD x = 2.0;
7    FAD y = 3.0;
8    x.diff(0,2);
9    y.diff(1,2);
10   std::cout << x << std::endl;
11   std::cout << y << std::endl;
12   std::cout << f(x,y) << std::endl;
13   std::cout << g(x,y) << std::endl;
14   std::cout << f(x,y) * g(x,y) << std::endl;
15 }
```

Line 1 includes the Sacado header, which is an automatic differentiation library developed by Sandia National Laboratories. Line 2 declares the forward AD type to be called

FAD. Line 3 declares f to be the sum of two AD variables x and y. Line 4 declares g to be the product of two AD variables x and y. Line 10 initializes the AD variable x to a value of 2 and line 12 *seeds* x to be the 1st variable in a system of 2 independent variables. Line 11 initializes the AD variable y to a value of 3 and line 13 *seeds* y to be the 2nd variable in a system of 2 independent variables. Lines 14-18 print the results of evaluating various functions.

The output from this simple example program is shown below

```
x:      2 [ 1 0 ]
y:      3 [ 0 1 ]
f:      5 [ 1 1 ]
g:      6 [ 3 2 ]
f*g:    30 [ 21 16 ]
```

Naturally, the value of $x = 2$ and $y = 3$, as we initialized them to be. Similarly, the derivative array of x shows that $\frac{\partial x}{\partial x} = 1$ and $\frac{\partial x}{\partial y} = 0$ and the derivative array of y shows that $\frac{\partial y}{\partial x} = 0$ and $\frac{\partial y}{\partial y} = 1$.

The value of $f(x, y) = z + y$ is given as $2 + 3 = 5$, and its derivative array is updated according to operator overloading of the $+$ operator as the sum of x's derivative array and y's derivative array:

$$\boldsymbol{f'} = \boldsymbol{x'} + \boldsymbol{y'} = [1\ 0] + [0\ 1] = [1\ 1]. \tag{A.4}$$

The value of $g(x, y) = x \cdot y$ is given as $2 \cdot 3 = 6$, and its derivative array is updated according to operator overloading of the $*$ operator as the sum of $\boldsymbol{x'}$ times y and $\boldsymbol{y'}$ times x:

$$\boldsymbol{g'} = \boldsymbol{x'} \cdot y + x \cdot \boldsymbol{y'} = [1\ 0] \cdot 3 + 2 \cdot [0\ 1] = [3\ 2]. \tag{A.5}$$

Finally, the value of $h(x, y) = f(x, y) \cdot g(x, y)$ is computed as $6 * 5 = 30$, and its derivative array is computed by propogating $\boldsymbol{f'}$ and $\boldsymbol{g'}$ through the code as:

$$\boldsymbol{h'} = \boldsymbol{f'} \cdot g + f \cdot \boldsymbol{g'} = [1\ 1] \cdot 6 + 5 \cdot [3\ 2] = [21\ 16]. \tag{A.6}$$

APPENDIX B
PROPOSITIONS FOR THE ADVECTION-DIFFUSION OPERATOR

B.1 Non-Homogeneous Boundary Conditions

Extensions to non-homogeneous Dirichlet boundary conditions for the primal model, given as

$$\begin{cases} \mathcal{L}u = f, & \boldsymbol{x} \in \Omega, \\ u = g, & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{B.1}$$

can readily be made by introducing the decomposition $u = u_0 + \widetilde{g}$, where $\mathrm{tr}(\widetilde{g}) = g$. The problem is then reposed as a homogeneous Dirichlet problem given by

$$\begin{cases} \mathcal{L}u_0 = f + \mathcal{L}\widetilde{g}, & \boldsymbol{x} \in \Omega, \\ u_0 = 0, & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{B.2}$$

where all arguments made previously can be applied to this modified formulation, provided $f + \mathcal{L}\widetilde{g} \in \mathcal{V}^*$.

Extensions to non-homogeneous Neumann boundary conditions require additional investigation. To proceed, consider the primal problem given as

$$\begin{cases} \mathcal{L}u = f & \boldsymbol{x} \in \Omega, \\ u = 0, & \boldsymbol{x} \in \partial\Omega_D \\ Bu = h & \boldsymbol{x} \in \partial\Omega_N, \end{cases} \tag{B.3}$$

where $\Omega_D \cup \Omega_N = \Omega$ and $\Omega_D \cap \Omega_N = \{\varnothing\}$. Multiplying the left hand side of the primal problem (B.3) by an arbitrary test function v and integrating by parts over the domain

This chapter previously appeared as: B. N. Granzow, M. S. Shephard, and A. A. Oberai, "Output-based error estimation and mesh adaptation for variational multiscale methods." *Comput. Methods in Appl. Mechanics and Eng.*, vol. 322, pp. 331-459, Aug. 2017.

twice yields the relationship

$$\int_{\Omega} v\mathcal{L}u \,\mathrm{d}\Omega + \int_{\partial\Omega_N} vBu \,\mathrm{d}\Gamma = \int_{\Omega} \mathcal{L}^* vu \,\mathrm{d}\Omega + \int_{\partial\Omega_N} B^* vu \,\mathrm{d}\Gamma. \tag{B.4}$$

All subsequent derivations would need to be made considering this relationship, which involves the boundary operator B, rather than relationship (4.9) which has been used extensively in this paper.

B.2 Derivation of the Advection-Diffusion Adjoint Operator

Let $\mathcal{L} : \mathcal{V} \to \mathcal{V}^*$ be the steady-state, constant coefficient operator utilized in section 4.6:

$$\mathcal{L}u := -\kappa\nabla^2 u + \boldsymbol{a} \cdot \nabla u, \tag{B.5}$$

such that $\mathcal{V} = H_0^1(\Omega)$ and $\mathcal{V}^* = H^{-1}(\Omega)$. To determine the corresponding operator: $\mathcal{L}^*$ that satisfies the adjoint property:

$$_{\mathcal{V}^*}\langle z, \mathcal{L}u\rangle_{\mathcal{V}} = \, _{\mathcal{V}}\langle \mathcal{L}^* z, u\rangle_{\mathcal{V}^*} \quad \forall\, u, z \in H_0^1(\Omega), \tag{B.6}$$

we multiply $\mathcal{L}u$ by an arbitrary function $z \in H_0^1(\Omega)$ and repeatedly apply the divergence theorem. This proceeds as follows:

$$
\begin{aligned}
{}_{\mathcal{V}}\langle z, \mathcal{L}u\rangle_{\mathcal{V}^*} &= \int_\Omega z(-\kappa\nabla^2 u + \boldsymbol{a}\cdot\nabla u)\,\mathrm{d}\Omega \\
&= -\int_\Omega z\kappa\nabla^2 u\,\mathrm{d}\Omega + \int_\Omega z\boldsymbol{a}\cdot\nabla u\,\mathrm{d}\Omega \\
&= -\int_{\partial\Omega} z\kappa\nabla u\cdot\boldsymbol{n}\,\mathrm{d}\Gamma + \int_\Omega \kappa\nabla z\cdot\nabla u\,\mathrm{d}\Omega + \\
&\qquad \int_{\partial\Omega}(z\boldsymbol{a}u)\cdot\boldsymbol{n}\,\mathrm{d}\Gamma - \int_\Omega \boldsymbol{a}\cdot\nabla z u\,\mathrm{d}\Omega \\
&= \int_\Omega \kappa\nabla z\cdot\nabla u\,\mathrm{d}\Omega - \int_\Omega \boldsymbol{a}\cdot\nabla z u\,\mathrm{d}\Omega \\
&= \int_{\partial\Omega}(\kappa\nabla z u)\cdot\boldsymbol{n}\,\mathrm{d}\Gamma - \int_\Omega \kappa\nabla^2 z u\,\mathrm{d}\Omega - \int_\Omega \boldsymbol{a}\cdot\nabla z u\,\mathrm{d}\Omega \\
&= -\int_\Omega \kappa\nabla^2 z u\,\mathrm{d}\Omega - \int_\Omega \boldsymbol{a}\cdot\nabla z u\,\mathrm{d}\Omega \\
&= \int_\Omega(-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)u\,\mathrm{d}\Omega \\
&= {}_{\mathcal{V}^*}\langle\mathcal{L}^* z, u\rangle_{\mathcal{V}}\,.
\end{aligned}
$$

Here the third equality is achieved by application of the divergence theorem to both terms, the fourth equality holds since $z \in H_0^1(\Omega)$, the fifth equality is achieved by application of the divergence theorem to the leftmost term, and the sixth equality holds since $u \in H_0^1(\Omega)$. Thus, the operator $\mathcal{L}^* : H_0^1(\Omega) \to H^{-1}(\Omega)$ is defined as

$$
\mathcal{L}^* z := -\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z. \tag{B.7}
$$

We make the observation that there has been a sign change for the advective term since the operator $\mathcal{L}$ is not self-adjoint. This sign change, however, is absorbed in the definition of the operator $\mathcal{L}^*$ and in no way introduces a sign change in the fundamental property:

$$
{}_{\mathcal{V}}\langle z, \mathcal{L}u\rangle_{\mathcal{V}^*} = {}_{\mathcal{V}^*}\langle\mathcal{L}^* z, u\rangle_{\mathcal{V}} \quad \forall\, u, z \in H_0^1(\Omega). \tag{B.8}
$$

B.3 Propositions Applied to the Advection-Diffusion Operator

We restate the adjoint property (B.8) as

$$\int_\Omega z(-\kappa\nabla^2 u + \boldsymbol{a}\cdot\nabla u)\,\mathrm{d}\Omega = \int_\Omega (-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)u\,\mathrm{d}\Omega \quad \forall\, u, z \in H_0^1(\Omega). \tag{B.9}$$

We now define the primal problem as:

$$\begin{cases} -\kappa\nabla^2 u + \boldsymbol{a}\cdot\nabla u = f, & \boldsymbol{x}\in\Omega, \\ u = 0, & \boldsymbol{x}\in\partial\Omega, \end{cases} \tag{B.10}$$

corresponding to equation (4.1), where $f \in H^{-1}(\Omega)$. We note that the primal residual operator is given as:

$$\mathcal{R}u := f + \kappa\nabla^2 u - \boldsymbol{a}\cdot\nabla u. \tag{B.11}$$

We define the continuous variational multiscale formulation of the primal problem as: find $\overline{u} \in \overline{\mathcal{V}}$ such that

$$\int_\Omega (-\kappa\nabla^2\overline{v} - \boldsymbol{a}\cdot\nabla\overline{v})\mathcal{G}'\mathcal{R}\overline{u}\,\mathrm{d}\Omega = \int_\Omega \overline{v}\mathcal{R}\overline{u}\,\mathrm{d}\Omega \quad \forall\,\overline{v}\in H_0^1(\Omega), \tag{B.12}$$

corresponding to equation (4.11), where we leave the fine-scale Green's operator $\mathcal{G}' : H^{-1}(\Omega) \to H_0^1(\Omega)$ as an unspecified abstract operator. Here we note that $\mathcal{G}'\mathcal{R}\overline{u} \in H_0^1(\Omega)$. Let $\mathcal{V}^h \subset \mathcal{V}$ denote a classical finite element space consisting of piecewise linear functions defined over a discretization of the domain Ω. The primal subgrid model can then be stated as: find $u^h \in \mathcal{V}^h$ such that

$$\sum_{e=1}^{n_{el}} \int_{\Omega^e} (-\kappa\nabla^2 v^h - \boldsymbol{a}\cdot\nabla v^h)(\tau^e\mathcal{R}u^h)\,\mathrm{d}\Omega = \int_\Omega v^h\mathcal{R}u^h\,\mathrm{d}\Omega \quad \forall\, v^h \in \mathcal{V}^h, \tag{B.13}$$

corresponding to equation (4.15), where we leave τ^e unspecified.

We define the dual problem as:

$$\begin{cases} -\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z = q, & \boldsymbol{x}\in\Omega, \\ z = 0, & \boldsymbol{x}\in\partial\Omega, \end{cases} \tag{B.14}$$

corresponding to equation (4.23), where $q \in H^{-1}(\Omega)$. We note that the dual residual operator is given as:

$$\mathcal{R}^* z := q + \kappa \nabla^2 z + \boldsymbol{a} \cdot \nabla z. \tag{B.15}$$

We define the continuous variational multiscale formulation of the dual problem as: find $\overline{z} \in \overline{\mathcal{V}}$ such that

$$\int_\Omega \mathcal{G}'_d \mathcal{R}^* \overline{z} (-\kappa \nabla^2 \overline{v} + \boldsymbol{a} \cdot \nabla \overline{v}) \, \mathrm{d}\Omega = \int_\Omega \mathcal{R}^* \overline{z} \overline{v} \, \mathrm{d}\Omega \quad \forall \, \overline{v} \in H_0^1(\Omega), \tag{B.16}$$

corresponding to equation (4.27), where again we leave the dual fine-scale Green's operator $\mathcal{G}'_d : H^{-1}(\Omega) \to H_0^1(\Omega)$ unspecified. We note that $\mathcal{G}'_d \mathcal{R}^* \overline{z} \in H_0^1(\Omega)$ The dual subgrid model can then be stated as: find $z^h \in \mathcal{V}^h$ such that

$$\sum_{e=1}^{n_{el}} \int_{\Omega^e} (\tau_d^e \mathcal{R}^* z^h)(-\kappa \nabla^2 v^h + \boldsymbol{a} \cdot \nabla v^h) \, \mathrm{d}\Omega = \int_\Omega \mathcal{R}^* z^h v^h \, \mathrm{d}\Omega \quad \forall \, v^h \in \mathcal{V}^h, \tag{B.17}$$

corresponding to equation (4.31), where we leave τ_d^e unspecified.

B.3.1 Proposition 2

For any solutions $u = u' + \overline{u}$ to the continuous VMS formulation (B.12) and $z = z' + \overline{z}$ to the continuous dual VMS formulation (B.16), we derive the error representation:

$$\begin{aligned}
J(u) - J(\overline{u}) &= \int_\Omega qu \,\mathrm{d}\Omega - \int_\Omega q\overline{u} \,\mathrm{d}\Omega \\
&= \int_\Omega (-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)u \,\mathrm{d}\Omega - \int_\Omega (-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)\overline{u} \,\mathrm{d}\Omega \\
&= \int_\Omega z(-\kappa\nabla^2 u + \boldsymbol{a}\cdot\nabla u) \,\mathrm{d}\Omega - \int_\Omega z(-\kappa\nabla^2 \overline{u} + \boldsymbol{a}\cdot\nabla \overline{u}) \,\mathrm{d}\Omega \\
&= \int_\Omega zf \,\mathrm{d}\Omega - \int_\Omega z(-\kappa\nabla^2 \overline{u} + \boldsymbol{a}\cdot\nabla \overline{u}) \,\mathrm{d}\Omega \\
&= \int_\Omega z\mathcal{R}\overline{u} \,\mathrm{d}\Omega \\
&= \int_\Omega z'\mathcal{R}\overline{u} \,\mathrm{d}\Omega + \int_\Omega \overline{z}\mathcal{R}\overline{u} \,\mathrm{d}\Omega \\
&= \int_\Omega z'\mathcal{R}\overline{u} \,\mathrm{d}\Omega + \int_\Omega (-\kappa\nabla^2 \overline{z} - \boldsymbol{a}\cdot\nabla \overline{z})\mathcal{G}'\mathcal{R}\overline{u} \,\mathrm{d}\Omega \\
&= \int_\Omega (\mathcal{G}_d'\mathcal{R}^*\overline{z})\mathcal{R}\overline{u} \,\mathrm{d}\Omega + \int_\Omega (-\kappa\nabla^2 \overline{z} - \boldsymbol{a}\cdot\nabla \overline{z})\mathcal{G}'\mathcal{R}\overline{u} \,\mathrm{d}\Omega \\
&= {}_{\mathcal{V}}\langle \mathcal{G}_d'\mathcal{R}^*\overline{z}, \mathcal{R}\overline{u}\rangle_{\mathcal{V}^*} + {}_{\mathcal{V}^*}\langle \mathcal{L}^*\overline{z}, \mathcal{G}'\mathcal{R}\overline{u}\rangle_{\mathcal{V}} \,.
\end{aligned}$$

Here the first equality is by definition (4.21), the second equality is due to the dual PDE (B.14), the third equality is due to the fundamental relation (B.9), the fourth equality is due to the primal PDE (B.10), the fifth equality is due to the definition of the primal residual (B.11), the sixth equality is due to the sum decomposition of the dual solution $z = z' + \overline{z}$, the seventh equality is due to the continuous variational formulation of the primal problem (B.12), the eight equality is due to the definition of the fine-scale dual solution (4.24), and the ninth equality is due to the definition of the duality pairing we have chosen.

B.3.2 Proposition 4

For any solutions u to the primal model (4.2), z to the dual model (4.22), u^h to the primal subgrid model (4.14) and z^h to the dual subgrid model (4.30), we derive the error

representation

$$
\begin{aligned}
J(u) - J(u^h) &= \int_\Omega qu \,\mathrm{d}\Omega - \int_\Omega qu^h \,\mathrm{d}\Omega \\
&= \int_\Omega (-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)u \,\mathrm{d}\Omega - \int_\Omega (-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)u^h \,\mathrm{d}\Omega \\
&= \int_\Omega z(-\kappa\nabla^2 u + \boldsymbol{a}\cdot\nabla u) \,\mathrm{d}\Omega - \int_\Omega z(-\kappa\nabla^2 u^h + \boldsymbol{a}\cdot\nabla u^h) \,\mathrm{d}\Omega \\
&= \int_\Omega zf \,\mathrm{d}\Omega - \int_\Omega z(-\kappa\nabla^2 u^h + \boldsymbol{a}\cdot\nabla u^h) \,\mathrm{d}\Omega \\
&= \int_\Omega z\mathcal{R}u^h \,\mathrm{d}\Omega \\
&= \int_\Omega z\mathcal{R}u^h \,\mathrm{d}\Omega - \int_\Omega z^h\mathcal{R}u^h \,\mathrm{d}\Omega + \\
&\qquad \sum_{e=1}^{n_{el}} \int_\Omega (-\kappa\nabla^2 z^h - \boldsymbol{a}\cdot\nabla z^h)(\tau^e\mathcal{R}u^h) \,\mathrm{d}\Omega \\
&= \int_\Omega (z - z^h)\mathcal{R}u^h + \sum_{e=1}^{n_{el}} \int_\Omega (-\kappa\nabla^2 z^h - \boldsymbol{a}\cdot\nabla z^h)(\tau^e\mathcal{R}u^h) \,\mathrm{d}\Omega \\
&= \int_\Omega (\widetilde{z}' + \widetilde{z})\mathcal{R}u^h + \sum_{e=1}^{n_{el}} \int_\Omega (-\kappa\nabla^2 z^h - \boldsymbol{a}\cdot\nabla z^h)(\tau^e\mathcal{R}u^h) \,\mathrm{d}\Omega \\
&= \int_\Omega \widetilde{z}'\mathcal{R}u^h + \sum_{e=1}^{n_{el}} \int_\Omega (-\kappa\nabla^2 z^h - \boldsymbol{a}\cdot\nabla z^h)(\tau^e\mathcal{R}u^h) \,\mathrm{d}\Omega + \\
&\qquad \int_\Omega \widetilde{z}\mathcal{R}u^h \\
&= \int_\Omega (\tau_d^e\mathcal{R}^* z^h)\mathcal{R}u^h + \sum_{e=1}^{n_{el}} \int_\Omega (-\kappa\nabla^2 z^h - \boldsymbol{a}\cdot\nabla z^h)(\tau^e\mathcal{R}u^h) \,\mathrm{d}\Omega + \\
&\qquad \int_\Omega \widetilde{z}\mathcal{R}u^h,
\end{aligned}
$$

where the first equality is by definition (4.21), the second equality is due to the dual PDE (B.14), the third equality is due to the fundamental relationship (B.9), the fourth equality is due to the primal PDE (B.10), the fifth equality is due to the definition of the primal residual (B.11), the sixth equality is due to the primal subgrid model (B.13) (where we have added and subtracted equal terms), the seventh equality is due to linearity, the eighth equality is due to the decomposition of the dual solution (4.19), the ninth equality is due to linearity, and the tenth equality is due to the fine-scale approximation to the dual solution (4.32).

B.3.3 Proposition 5

We first note that the derivation in B.2 can be carried out in exactly the same manner for $u, z \in H_0^1(\Omega^e)$ to obtain the result:

$$\begin{aligned} &\int_{\Omega^e} z(-\kappa\nabla^2 u + \boldsymbol{a}\cdot\nabla u)\,\mathrm{d}\Omega = \\ &\int_{\Omega^e} (-\kappa\nabla^2 z - \boldsymbol{a}\cdot\nabla z)u\,\mathrm{d}\Omega \quad \forall\, u, z \in H_0^1(\Omega^e). \end{aligned} \tag{B.18}$$

We note that the problem (4.12) defining the primal element-level Green's function implies that $g^e(\boldsymbol{x};\boldsymbol{y}) \in H_0^1(\Omega^e)$. Similarly, the dual element-level Green's function satisfies $g_d^e(\boldsymbol{x};\boldsymbol{y}) \in H_0^1(\Omega^e)$ from equation (4.28). With this information, we utilize the relationship (B.18) to verify that $g^e(\boldsymbol{x};\boldsymbol{y}) = g_d^e(\boldsymbol{x};\boldsymbol{y})$, even though the operator $\mathcal{L}$ is not self-adjoint.

$$\begin{aligned} &-\kappa\nabla^2 g^e(\boldsymbol{x};\boldsymbol{y}) - \boldsymbol{a}\cdot\nabla g^e(\boldsymbol{x};\boldsymbol{y}) = \delta(\boldsymbol{x}-\boldsymbol{y}) \\ \Longrightarrow &\int_{\Omega^e} g_d^e(\boldsymbol{x};\boldsymbol{z})(-\kappa\nabla^2 g^e(\boldsymbol{x};\boldsymbol{y}) - \boldsymbol{a}\cdot\nabla g^e(\boldsymbol{x};\boldsymbol{y}))\,\mathrm{d}\Omega = \\ &\qquad \int_{\Omega^e} g_d^e(\boldsymbol{x};\boldsymbol{z})\delta(\boldsymbol{x}-\boldsymbol{y})\,\mathrm{d}\Omega \\ \Longrightarrow &\int_{\Omega^e} (-\kappa\nabla^2 g_d^e(\boldsymbol{x};\boldsymbol{z}) + \boldsymbol{a}\cdot\nabla g_d^e(\boldsymbol{x};\boldsymbol{z}))g^e(\boldsymbol{x};\boldsymbol{y})\,\mathrm{d}\Omega = \\ &\qquad \int_{\Omega^e} g_d^e(\boldsymbol{x};\boldsymbol{z})\delta(\boldsymbol{x}-\boldsymbol{y})\,\mathrm{d}\Omega \\ \Longrightarrow &\int_{\Omega^e} \delta(\boldsymbol{x}-\boldsymbol{z})g^e(\boldsymbol{x};\boldsymbol{y})\,\mathrm{d}\Omega = \int_{\Omega^e} g_d^e(\boldsymbol{x};\boldsymbol{z})\delta(\boldsymbol{x}-\boldsymbol{y})\,\mathrm{d}\Omega \\ \Longrightarrow &\, g^e(\boldsymbol{z};\boldsymbol{y}) = g_d^e(\boldsymbol{y};\boldsymbol{z}), \end{aligned}$$

Here the first equality is due to the definition of the primal element-level Green's function (4.12), the second equality is achieved by multiplying by the dual element-level Green's function and integrating over the element domain, the third equality is due to the fundamental relationship (B.18), and the fourth equality is due to the definition of the dual element-level Green's function (4.28).